Model*Sim* EE/SE

Tutorial

Version 5.3

The Model*Sim* Elite and Special Editions

for VHDL, Verilog, and Mixed-HDL Simulation

Model*Sim* /VHDL, Model*Sim* /VLOG, Model*Sim* /LNL, and Model*Sim* /PLUS are produced by Model Technology Incorporated. Unauthorized copying, duplication, or other reproduction is prohibited without the written consent of Model Technology.

The information in this manual is subject to change without notice and does not represent a commitment on the part of Model Technology. The program described in this manual is furnished under a license agreement and may not be used or copied except in accordance with the terms of the agreement. The online documentation provided with this product may be printed by the end-user. The number or copies that may be printed is limited to the number of licenses purchased.

Model*Sim* is a trademark of Model Technology Incorporated. PostScript is a registered trademark of Adobe Systems Incorporated. UNIX is a registered trademark of AT&T in the USA and other countries. FLEXlm is a trademark of Globetrotter Software, Inc. IBM, AT, and PC are registered trademarks, AIX and RISC System/6000 are trademarks of International Business Machines Corporation. Windows, Microsoft, and MS-DOS are registered trademarks of Microsoft Corporation. OSF/Motif is a trademark of the Open Software Foundation, Inc. in the USA and other countries. SPARC is a registered trademark and SPARCstation is a trademark of SPARC International, Inc. Sun Microsystems is a registered trademark, and Sun, SunOS and OpenWindows are trademarks of Sun Microsystems, Inc. All other trademarks and registered trademarks are the properties of their respective holders.

Copyright (c) 1990 -1999, Model Technology Incorporated.
All rights reserved. Confidential. Online documentation may be printed by licensed customers of Model Technology Incorporated for internal business purposes only.

Software Version: 5.3

Published: August 1999
Model Technology Incorporated
10450 SW Nimbus Avenue / Bldg. R-B
Portland OR 97223-4347 USA

phone: 503-641-1340
fax: 503-526-5410
e-mail: support@model.com, sales@model.com
home page: http://www.model.com

EE Tutorial - Part # M16540 US$50

Software License Agreement

This is a legal agreement between you, the end user, and Model Technology Incorporated (MTI). By opening the sealed package you are agreeing to be bound by the terms of this agreement. If you do not agree to the terms of this agreement, promptly return the unopened package and all accompanying items to the place you obtained them for a full refund.

Model Technology Software License

1. LICENSE. MTI grants to you the **nontransferable**, **nonexclusive** right to use one copy of the enclosed software program (the "SOFTWARE") for each license you have purchased. The SOFTWARE must be used on the computer hardware server equipment that you identified in writing by make, model, and workstation or host identification number and the equipment served, in machine-readable form only, as allowed by the authorization code provided to you by MTI or its agents. All authorized systems must be used within the country for which the systems were sold. Model*Sim* licenses must be located at a single site, i.e. within a one-kilometer radius identified in writing to MTI. This restriction does not apply to single ModelSim PE licenses locked by a hardware security key, and such Model*Sim* PE products may be relocated within the country for which sold.

2. COPYRIGHT. The SOFTWARE is owned by MTI (or its licensors) and is protected by United States copyright laws and international treaty provisions. Therefore you must treat the SOFTWARE like any other copyrighted material, except that you may either (a) make one copy of the SOFTWARE solely for backup or archival purposes, or (b) transfer the SOFTWARE to a single hard disk provided you keep the original solely for backup or archival purposes. You may not copy the written materials accompanying the SOFTWARE.

3. USE OF SOFTWARE. The SOFTWARE is licensed to you for internal use only. You shall not conduct benchmarks or other evaluations of the SOFTWARE without the advance written consent of an authorized representative of MTI. You shall not sub-license, assign or otherwise transfer the license granted or the rights under it without the prior written consent of MTI or its applicable licensor. You shall keep the SOFTWARE in a restricted and secured area and shall grant access only to authorized persons. You shall not make software available in any form to any person other than your employees whose job performance requires access and who are specified in writing to MTI. MTI may enter your business premises during normal business hours to inspect the SOFTWARE, subject to your normal security.

4. PERMISSION TO COPY LICENSED SOFTWARE. You may copy the SOFTWARE only as reasonably necessary to support an authorized use. Except as permitted by Section 2, you may not make copies, in whole or in part, of the SOFTWARE or other material provided by MTI without the prior written consent of MTI. For such permitted copies, you will include all notices and legends embedded in the SOFTWARE and affixed to its medium and container as received

from MTI. All copies of the SOFTWARE, whether provided by MTI or made by you, shall remain the property of MTI or its licensors.

You will maintain a record of the number and location of all copies of the SOFTWARE made, including copes that have been merged with other software, and will make those records available to MTI or its applicable licensor upon request.

5. TRADE SECRET. The source code of the SOFTWARE is trade secret or confidential information of MTI or its licensors. You shall take appropriate action to protect the confidentiality of the SOFTWARE and to ensure that any user permitted access to the SOFTWARE does not provide it to others. You shall take appropriate action to protect the confidentiality of the source code of the SOFTWARE. You shall not reverse-assemble, reverse-compile or otherwise reverse-engineer the SOFTWARE in whole or in part. The provisions of this section shall survive the termination of this Agreement.

6. TITLE. Title to the SOFTWARE licensed to you or copies thereof are retained by MTI or third parties from whom MTI has obtained a licensing right.

7. OTHER RESTRICTIONS. You may not rent or lease the SOFTWARE. You shall not mortgage, pledge or encumber the SOFTWARE in any way. You shall ensure that all support service is performed by MTI or its designated agents. You shall notify MTI of any loss of the SOFTWARE.

8. TERMINATION. MTI may terminate this Agreement, or any license granted under it, in the event of breach or default by you. In the event of such termination, all applicable SOFTWARE shall be returned to MTI or destroyed.

9. EXPORT. You agree not to allow the MTI SOFTWARE to be sent or used in any other country except in compliance with this license and applicable U.S. laws and regulations. If you need advice on export laws and regulations, you should contact the U.S. Department of Commerce, Export Division, Washington, DC 20230, USA for clarification.

Important Notice

Any provision of Model Technology Incorporated SOFTWARE to the U.S. Government is with "Restricted Rights" as follows: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraphs (a) through (d) of the Commercial Computer-Restricted Rights clause at FAR 2.227-19 when applicable, or in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clauses in the NASA FAR Supplement. Any provision of Model Technology documentation to the U.S. Government is with Limited Rights. Contractor/manufacturer is Model Technology Incorporated, 10450 SW Nimbus Avenue / Bldg. R, Portland, Oregon 97223 USA.

Limited Warranty

LIMITED WARRANTY. MTI warrants that the SOFTWARE will perform substantially in accordance with the accompanying written materials for a period of 30 days from the date of receipt. Any implied warranties on the SOFTWARE are limited to 30 days. Some states do not allow limitations on duration of an implied warranty, so the above limitation may not apply to you.

CUSTOMER REMEDIES. MTI's entire liability and your exclusive remedy shall be, at MTI's option, either (a) return of the price paid or (b) repair or replacement of the SOFTWARE that does not meet MTI's Limited Warranty and which is returned to MTI. This Limited Warranty is void if failure of the SOFTWARE has resulted from accident, abuse or misapplication. Any replacement SOFTWARE will be warranted for the remainder of the original warranty period or 30 days, whichever is longer.

NO OTHER WARRANTIES. MTI disclaims all other warranties, either express or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose, with respect to the SOFTWARE and the accompanying written materials. This limited warranty gives you specific legal rights. You may have others, which vary from state to state.

NO LIABILITY FOR CONSEQUENTIAL DAMAGES. In no event shall MTI or its suppliers be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or other pecuniary loss) arising out of the use of or inability to use these MTI products, even if MTI has been advised of the possibility of such damages. Because some states do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation may not apply to you.

Table of Contents

Introduction 7

Before you begin 11

Lesson 1 - Creating a Project 13

Lesson 2 - Basic VHDL simulation 21

Lesson 3 - Debugging a VHDL design 31

Lesson 4 - Running a batch-mode simulation 41

Lesson 5 - Executing commands at startup 43

Lesson 6 - Tcl/Tk and ModelSim 45

Lesson 7 - Basic Verilog simulation 59

Lesson 8 - Mixed VHDL/Verilog simulation 73

Lesson 9 - Simulating with Performance Analyzer 81

Lesson 10 - Simulating with Code Coverage 93

Lesson 11 - Finding names, and searching for values 99

Lesson 12 - Using the Wave window 103

Index 111

Introduction

Chapter contents

Software versions 7
ModelSim's graphic interface 8
Standards supported 8
Assumptions 8
Where to find our documentation 9

The purpose of this tutorial is to help you become familiar with Model*Sim* EE, Model Technology's HDL-simulation software for UNIX, Microsoft Windows NT 4.0, and Windows 95/98. Lessons are included that apply to one or more of the following Model*Sim* simulators:

- **Model*Sim* EE/VHDL**
  for VHDL simulation only
- **Model*Sim* EE/VLOG**
  for Verilog simulation only
- **Model*Sim* EE/LNL**
  Language-Neutral Licensing - for either VHDL, **or** Verilog simulation
- **Model*Sim* EE/PLUS**
  for single language, or mixed VDHL/Verilog simulation

Software versions

This documentation was written to support ModelSim EE/SE 5.3 for UNIX, Microsoft Windows NT 4.0, and Windows 95/98. If the Model*Sim* software you are using is a later release, check the README file that accompanied the software. Any supplemental information will be there.

Although this document covers both VHDL and Verilog simulation, you will find it a useful reference even if your design work is limited to a single HDL.

Model*Sim*'s graphic interface

While your operating system interface provides the window-management frame, Model*Sim* controls all internal-window features including menus, buttons, and scroll bars. The resulting simulator interface remains consistent within these operating systems:

- SPARCstation with OpenWindows or OSF/Motif
- IBM RISC System/6000 with OSF/Motif
- Hewlett-Packard HP 9000 Series 700 with HP VUE or OSF/Motif
- Microsoft Windows NT and Windows 95/98

Because Model*Sim*'s graphic interface is based on Tcl/TK, you also have the tools to build your own simulation environment. Easily accessible preference variables and configuration commandsgive you control over the use and placement of windows, menus, menu options and buttons.

Standards supported

Model*Sim* VHDL supports the IEEE 1076-1987, 1076-1993 VHDL, 1164-1993 *Standard Multivalue Logic System for VHDL Interoperability* and the 1076.2-1996 *Standard VHDL Mathematical Packages* standards. Any design developed with Model*Sim* will be compatible with any other VHDL system that is compliant with either IEEE Standard 1076-1987 or 1076-1993.

Model*Sim* Verilog is based on the IEEE Std 1364-1995 *Standard Hardware Description Language Based on the Verilog Hardware Description Language*. (Model*Sim* 5.2e does not currently support VPI routines for PLI 2.0.) The Open Verilog International *Verilog LRM version 2.0* is also applicable to a large extent. Both PLI (Programming Language Interface) and VCD (Value Change Dump) are supported for ModelSim PE and EE users.

In addition, all products support SDF 1.0 through 3.0, VITAL 2.2b, and VITAL'95 - IEEE 1076.4-1995.

Assumptions

We assume that you are familiar with the use of your operating system. You should be familiar with the window management functions of your graphic interface: either OpenWindows, OSF/Motif, or Microsoft Windows NT/95/98.

We also assume that you have a working knowledge of VHDL and Verilog. Although Model*Sim* is an excellent tool to use while learning HDL concepts and practices, this document is not written to support that goal.

Comments

Comments and questions about this manual and Model*Sim* software are welcome. Call, write, or fax or email:

Model Technology Incorporated
10450 SW Nimbus Avenue, Bldg. R-B
Portland, OR 97223-4347 USA

phone: 503-641-1340
fax: 503-526-5410
email: manuals@model.com
home page: http://www.model.com

Where to find our documentation

Model*Sim* documentation is available from our website at model.com/resources/index.html or in the following formats and locations:

Document	Format	How to get it
Start Here for ModelSim EE (installation & support reference)	paper	shipped with Model*Sim*
	PDF online	from the Model*Sim* **Help** menu (select **EE Documentation > Licensing and Support**), or find *ee_start.pdf* in the *\modeltech\docs* directory; also available from the Support page of our web site: www.model.com
ModelSim EE Quick Guide (command and feature quick-reference)	paper	shipped with Model*Sim*
	PDF online	from the Model*Sim* **Help** menu (select **EE Documentation > EE Quick Guide**), or find *ee_guide.pdf* in the *\modeltech\docs* directory; also available from the Support page of our web site: www.model.com

Document	Format	How to get it
ModelSim EE Tutorial	PDF online	from the Model*Sim* **Help** menu in the **EE Documentation** group, or find *ee_tutor.pdf* in the */modeltech/docs* directory on the CD-ROM, or hard drive after installation, also available from the Support page of our web site: www.model.com
	paper	first two copies free with request cards in *Start Here* document; additional copies at $50 each (for customers with current maintenance)
ModelSim EE User's Manual	PDF online	from the Model*Sim* **Help** menu in the **EE Documentation** group, or find *ee_man.pdf* in the */modeltech/docs* directory on the CD-ROM, or hard drive after installation
	paper	first two copies free with request cards in *Start Here* document; additional copies at $50 each (for customers with current maintenance)
ModelSim EE Command Reference	PDF online	from the Model*Sim* **Help** menu in the **EE Documentation** group, or find *ee_cmds.pdf* in the */modeltech/docs* directory on the CD-ROM, or hard drive after installation
	paper	first two copies free with request cards in *Start Here* document; additional copies at $50 each (for customers with current maintenance)
Tcl man pages (Tcl manual)	HTML	use the Main window menu selection: **Help > Tcl Man Pages**, or find *contents.html* in *\modeltech\docs\html*
tech notes	ASCII	from the Model*Sim* Help menu, or located in the *\modeltech\docs\technotes* directory after installation

Before you begin

Preparation for some of the lessons leaves certain details up to you. You will decide the best way to create directories, copy files and execute programs within your operating system. (When you are operating the simulator within Model*Sim*'s GUI, the interface is consistent for all platforms.)

Additional details for VHDL, Verilog, and mixed VHDL/Verilog simulation can be found in the *ModelSim Reference Manual*. (See "Where to find our documentation" (p9).)

Examples show Windows path separators - use separators appropriate for your operating system when trying the examples.

Command, button, and menu equivalents

Many of the lesson steps are accomplished by a button or menu selection. When appropriate, VSIM command line (PROMPT:) or menu (MENU:) equivalents for these selections are shown in parentheses within the step. This example shows three options to the **run -all** command, a button, prompt command, and a menu selection.

(PROMPT: run -all) (MENU: Run > Run -All)

Drag and drop

Drag and drop allows you to copy and move signals among windows. If drag and drop applies to a lesson step, it is noted in a fashion similar to MENUS and PROMPTS with: DRAG&DROP.

Command history

As you work on the lessons keep an eye on the Main transcript. The commands invoked by buttons and menu selections are echoed there. You can scroll through the command history with the up and down arrow keys, or the command history may be reviewed with several shortcuts at the Model*Sim*/VSIM prompt.

Shortcut	Description
click on prompt	left-click once on a previous ModelSim or VSIM prompt in the transcript to copy the command typed at that prompt to the active cursor
`his or history`	shows the last few commands (up to 50 are kept)

Reusing commands from the Main transcript

ModelSim's Main transcript may be saved, and the resulting file used as a DO (macro) file to replay the transcribed commands. You can save the transcript at any time before or during simulation. You have the option of clearing the transcript (File > Clear Transcript) if you don't want to save the entire command history.

To save the contents of the transcript select **File > Save Main As** from the Main menu.

Replay the saved transcript with the **do** command:

```
do <do file name>
```

For example, if you saved a series of compiler commands as *mycompile.do* (the .do extension is optional), you could recompile with one command:

```
do mycompile.do
```

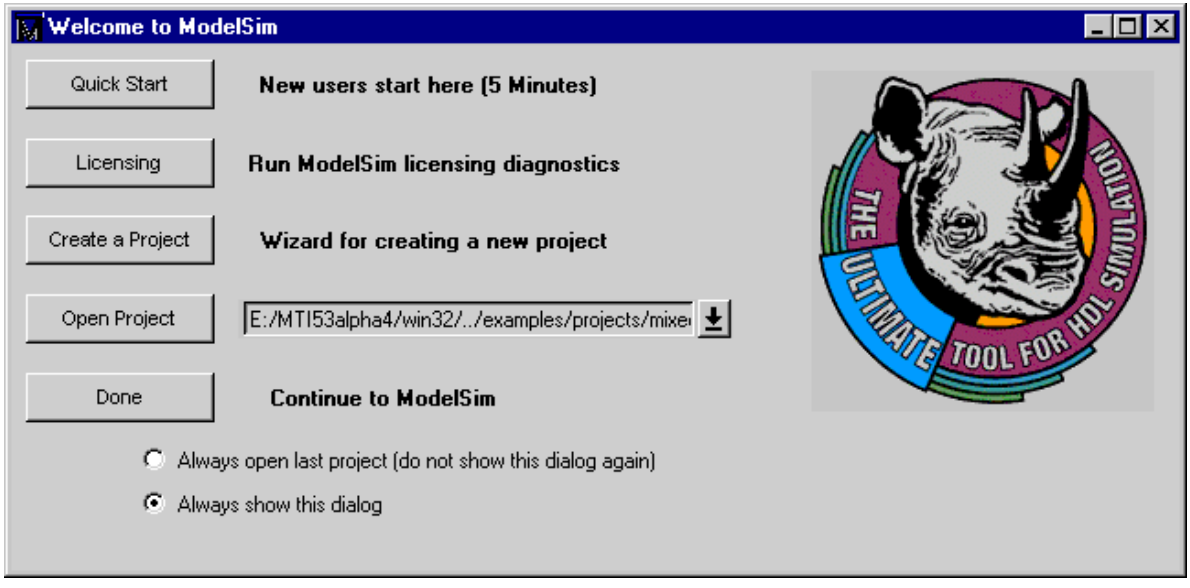
Note: Neither the prompt nor the Return that ends a command line are shown in the examples.

Lesson 1 - Creating a Project

The goals for the first lesson are:

- Explore the Welcome to ModelSim dialog box features.
- Create a project (.mpf file).

Upon opening Model*Sim* 5.3 for the first time, you will see a Welcome to Model*Sim* dialog box. (If this screen is not available, you can enable it by selecting **Help** > **Enable Welcome** from the Main window.)



From this dialog box, you can:

- View the Quick Start menus, which contain answers to commonly asked questions; or, click the See an Example button to get step by step instructions for creating projects (accessed with the).
- Run Model*Sim* licensing diagnostics.
- Use the project creation wizard to create a new project from scratch or to copy an existing project.
- Open an existing project.
- Click Done to continue to the Model*Sim* Main window.

Creating a Project

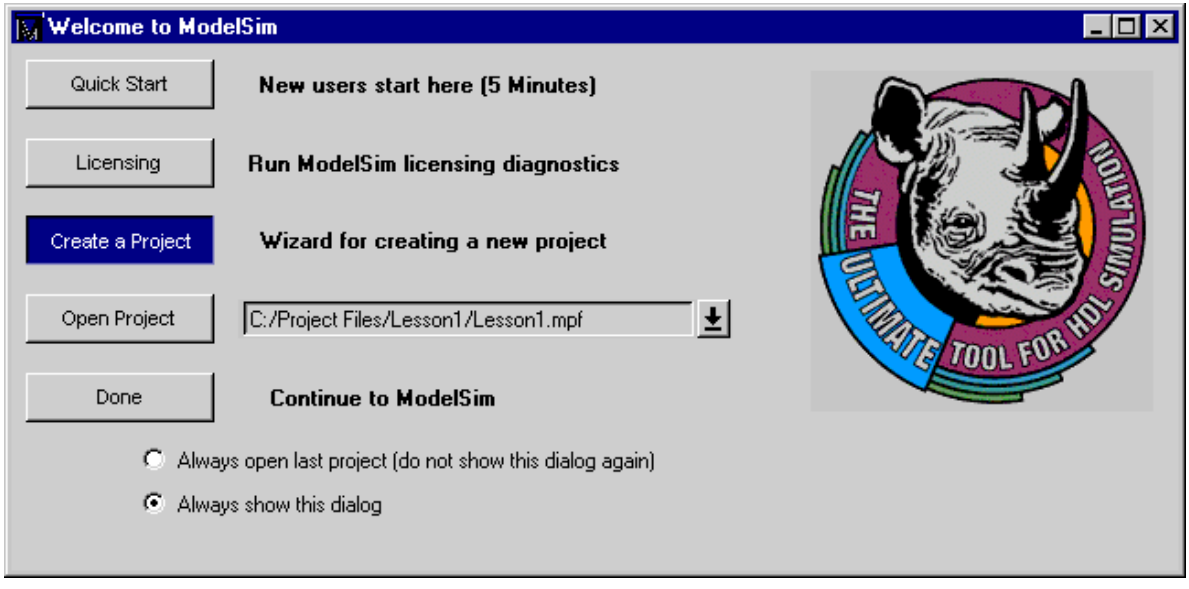
With the 5.3 release, Model*Sim* incorporates the file extension **.mpf** to denote project files. In past releases the *modelsim.ini* file (the system initialization file) was used as the project file.

A project is a collection entity for an HDL design under specification or test. At a minimum, it has a root directory, a work library and session state that is stored in a .mpf file located in the project's root directory. A project may also consist of:

- HDL source files
- subdirectories
- Local libraries
- References to global libraries

For more information about using project files, see the *ModelSim Reference Manual*.

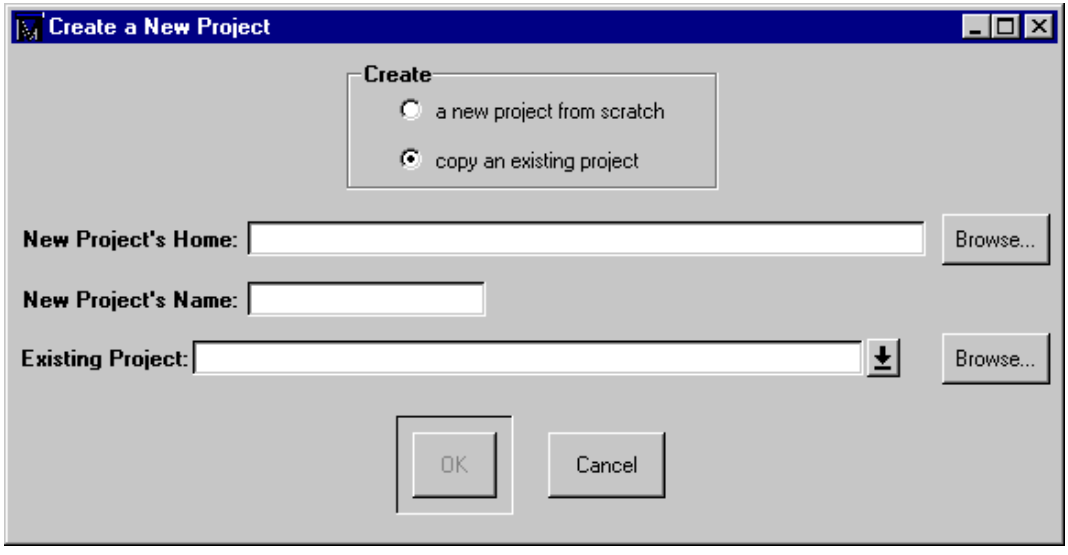
1 Select the **Create a Project** button from the Welcome to ModelSim dialog box.



Clicking the **Create a Project** button opens the Create a New Project dialog box and a project creation wizard. The wizard helps you through each step of creating a new project. It provides the option of entering Verilog or VHDL source file descriptions, then helps you load the project.

The Create a New Project dialog box can also be accessed by selecting **File > New > New Project** from the ModelSim Main window.

2 In the Create a New Project dialog box, you can elect to create a new project from scratch or copy an existing project. The easiest way to start any project is to copy a similar project. Let's begin by selecting "copy an existing project."

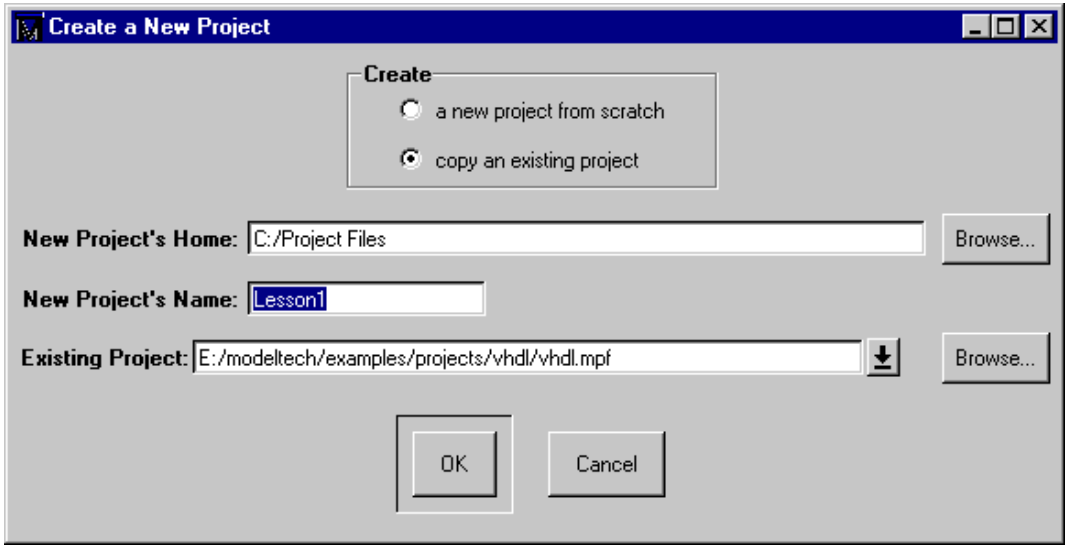


3 Specify a "New Project's Home," which is the directory under which the project's directory tree will reside. This directory must already exist – completion of this dialog box will not create it.

4 Specify the "New Project's Name" which will act as the project's directory name. It is recommended that a unique name be given to each project.

5 Specify an "Existing Project" name, which is the full path to an existing project's .mpf file. For this lesson, browse to *modeltech/examples/projects/vhdl/vhdl.mpf* and select this file.

Note: A project's .mpf file is always located in the project's directory.

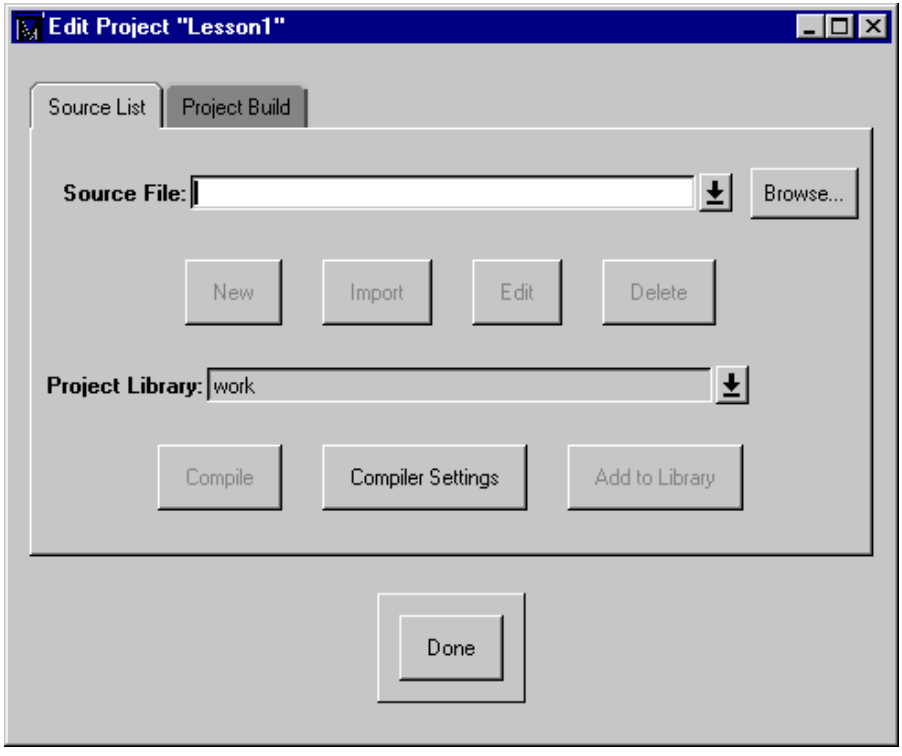


6 Once you have specified enough information to copy an existing project, the OK button is selectable. Selecting OK causes the project directory to be created with a default working library. You will then be asked if you want to make this project your current project. Click Yes.

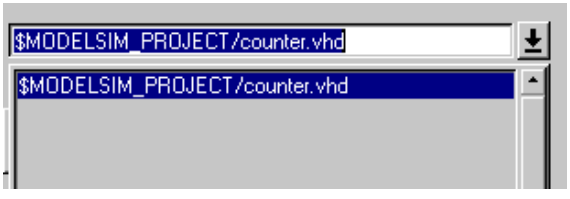
7 In the dialog box that asks if you want to create a new HDL source file for your project, click No. Then click Done in the Welcome to ModelSim dialog box.

No you can begin compiling your project.

8 Select **Options** > **Edit Project**. This opens the Edit Project dialog box.



Click the down arrow next to the Source File entry field and select the source *counter.vhd*.



With the source file selected, the Compile button becomes available.

9 Click the **Compile** button in the Edit Project dialog box. With a source file compiled, the Add to Library button becomes available.

10 Click **Add to Library**, then Done, in the Edit Project dialog box.

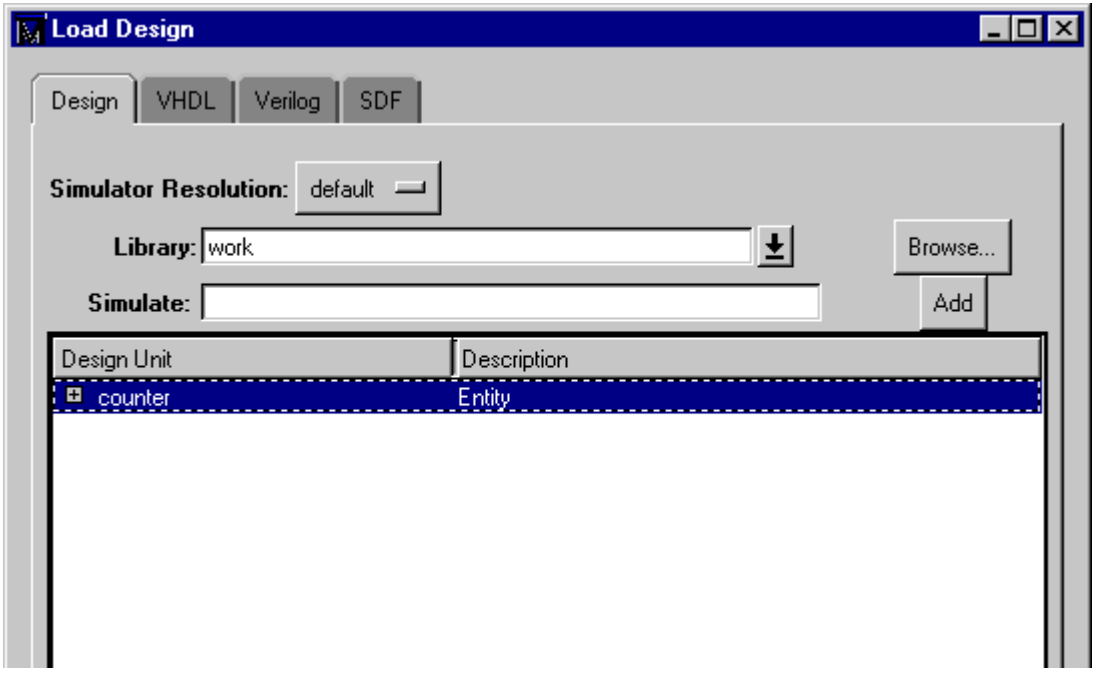
The Edit Project dialog also allows you to import a new source file into libraries local to the project.

Note: The process of **Compile** and **Add to Library** creates a script (DO file) that will recompile the entire project. You must **Add to Library** after each **Compile** to create a proper script. To recompile a project use the **Main > Design > Compile Project** menu selection.

11 Start the simulator by selecting the **Load Design** button from the toolbar:

(PROMPT: vsim counter)

The Load Design dialog box comes up, as shown below (you won't see this dialog box if you invoke **vsim** with *counter* from the command line).



The Load Design dialog box allows you to select the library and the top-level design unit to simulate.

This completes the process of creating a project by copying an existing project. The newly created project will be open for use in the Main window.

You can now elect to leave Model*Sim* or edit this project's HDL components until the project is completely specified and all files compile into libraries local to the project. If you should leave this Model*Sim* session, Model*Sim* will remember the last opened project so it can be reopened for your next session by simply clicking **Open Project** in the Welcome to Model*Sim* screen.

Lesson 2 - Basic VHDL simulation

The goals for this lesson are:

- Create a library.
- Compile a VHDL file.
- Start the simulator.
- Understand the basic VSIM windows, mouse, and menu conventions.
- Run VSIM using the **run** command.
- List some signals.
- Use the waveform display.
- Force the value of a signal.
- Single-step through a simulation run.
- Set a breakpoint.
- Use the Wave window.

1 Start out by creating a new directory for this exercise (in case other users will be working with these lessons). Create the directory, then copy all of the VHDL (.vhd) files from \<*install_dir*>*modeltech**examples* to the new directory.

Make sure the new directory is the current directory.

2 Start Model*Sim* with one of the following:

for UNIX at the shell prompt:

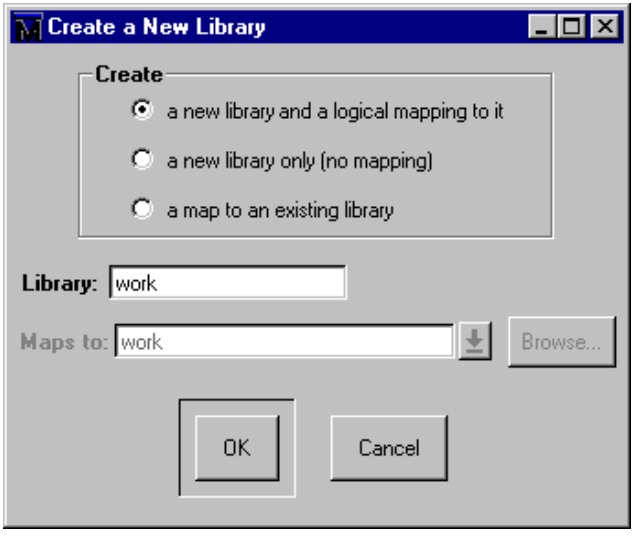
```
vsim -gui
```

for Windows - your option - from a Windows shortcut icon, from the Start menu, or from a DOS prompt:

```
modelsim.exe
```

This opens the Model*Sim* Main window and the Welcome to ModelSim dialog box. (This does not load a design.)

3 Before you compile any HDL code, you'll need a design library to hold the compilation results. To create a new design library, make this menu selection in the Main window: **Design > Create a New Library**.



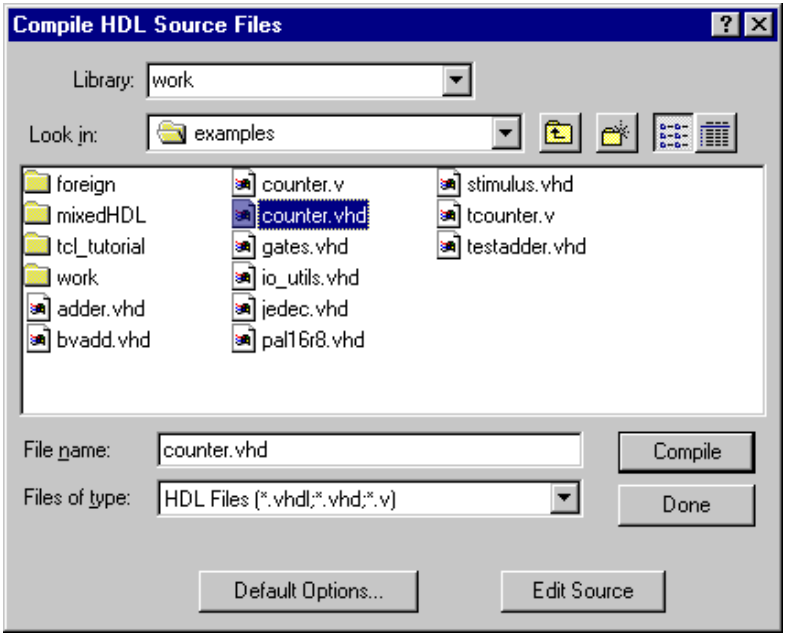
In the Create a New Library dialog box select **Create: a new library and a logical mapping to it**. Make sure **Library:** indicates **work**, then select **OK**. This creates a subdirectory named *work* - your design library - within the current directory. This subdirectory contains a special file named *_info*. (Do not create these using UNIX or Windows commands—always use the Library menu or the **vlib** command from either the Model*Sim* or UNIX/DOS prompt.)

(PROMPT: vlib work
vmap work work)

4 Compile the file *counter.vhd* into the new library by selecting the **Compile** button on the toolbar:

(PROMPT: vcom counter.vhd)

This opens the Compile HDL Source Files dialog box. (You won't see this dialog box if you invoke vcom from the command line.)



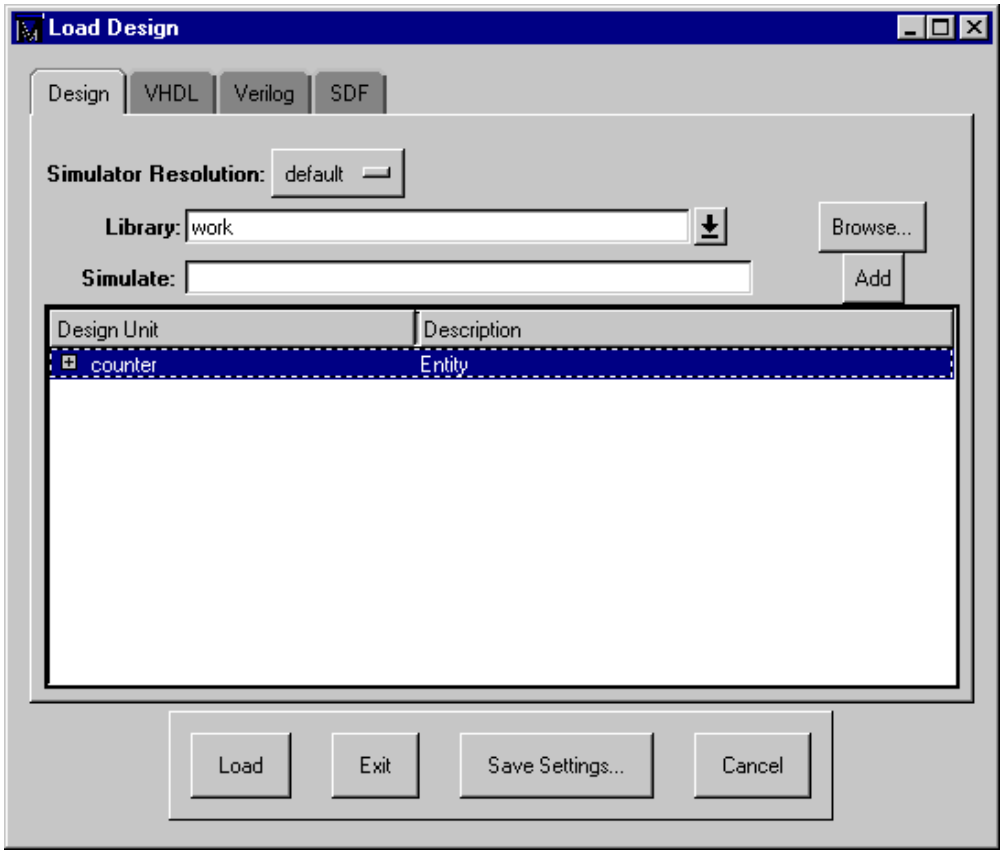
Complete the compilation by selecting *counter.vhd* from the file list and clicking **Compile**. Select **Done** when you are finished.

You can compile multiple files in one session from the file list. Individually select and Compile the files in the order required by your design.

5 Start the simulator by selecting the **Load Design** button from the toolbar:

(PROMPT: vsim counter)

The Load Design dialog box comes up, as shown below (you won't see this dialog box if you invoke **vsim** with *counter* from the command line).



The Load Design dialog box allows you to select the library and the top-level design unit to simulate. You can also select the resolution limit for this simulation. By default, the following will appear for this simulation run:

- Simulator Resolution: default (the default is 1 ns)
- Library: work
- Design Unit: counter
- Description: entity

Note: If the Design Unit is an entity (like **counter** in this design) you can click on the plus-box prefix to view any associated architectures; see the graphic below.

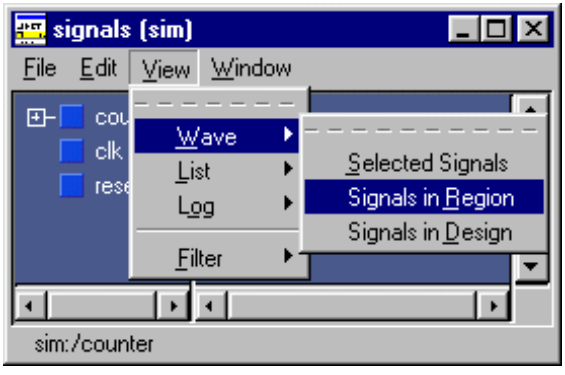
6 Select the entity **counter** and choose **Load** to accept these settings.

7 Now you can open all of the VSIM windows with this **Main window** menu selection: **View > All**.

(PROMPT: view *)

8 To display the top-level signals in the List window, select the **Signals window** and make this Signal menu selection: **View > List > Signals in Region**.

(PROMPT: add list /counter/*)



9 Next add top-level signals to the Wave window with a similar Signals menu selection: **View > Wave > Signals in Region**.

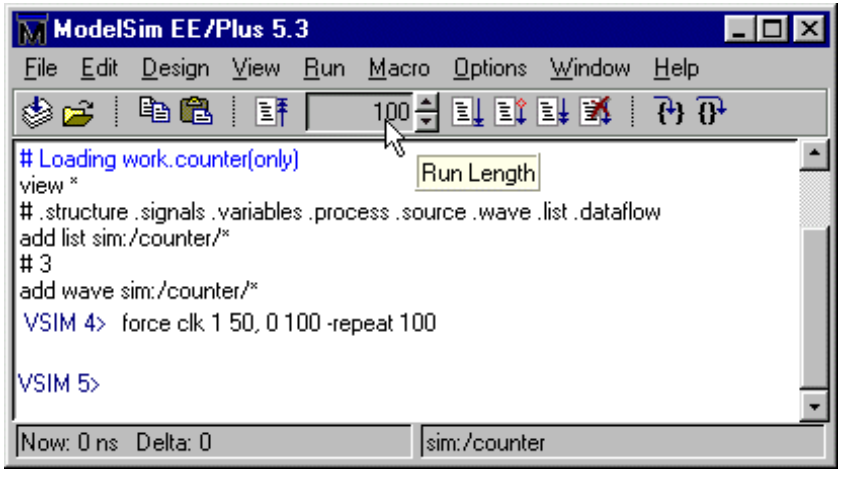
(PROMPT: add wave/counter/*)

10 You can apply stimulus to the clock input by moving the pointer to the **Main window** and entering the following command at the VSIM prompt:

```
force clk 1 50, 0 100 -repeat 100
```

The **force** command you just invoked is interpreted by VSIM to mean:

- force clk to the value 1 at 50 ns after the current time
- then to 0 at 100 ns after the current time
- repeat this cycle every 100 ns



Note how the Run Length selector on the toolbar now indicates 100 (ns is the current default resolution). You will see the effects of this **force** command as soon as you tell the simulator to run.

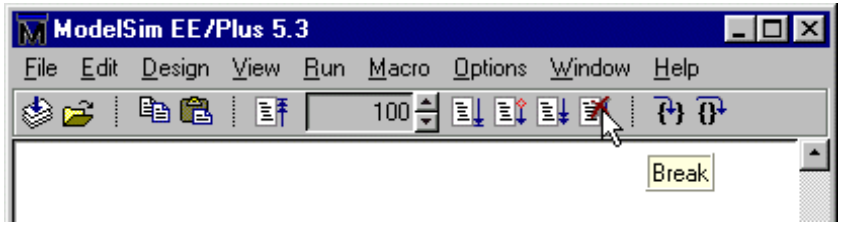
11 Now you will exercise two different **Run** functions from the toolbar buttons on either the Main or Wave window. (The **Run** functions are identical in the Main and Wave windows.) Select the **Run** button first. When the run is complete, select **Run All**.

Run. This causes the simulation to run and then stop after 100 ns.
(PROMPT: run 100) (MENU: Run > Run 100ns)

Run -All. This causes the simulator to run forever. To stop the run, go on to the next step.
(PROMPT: run -all) (MENU: Run > Run -All)

12 Select the **Break** button on either the **Main** or **Wave** toolbar to end the run.

(KEYBOARD: control+c)



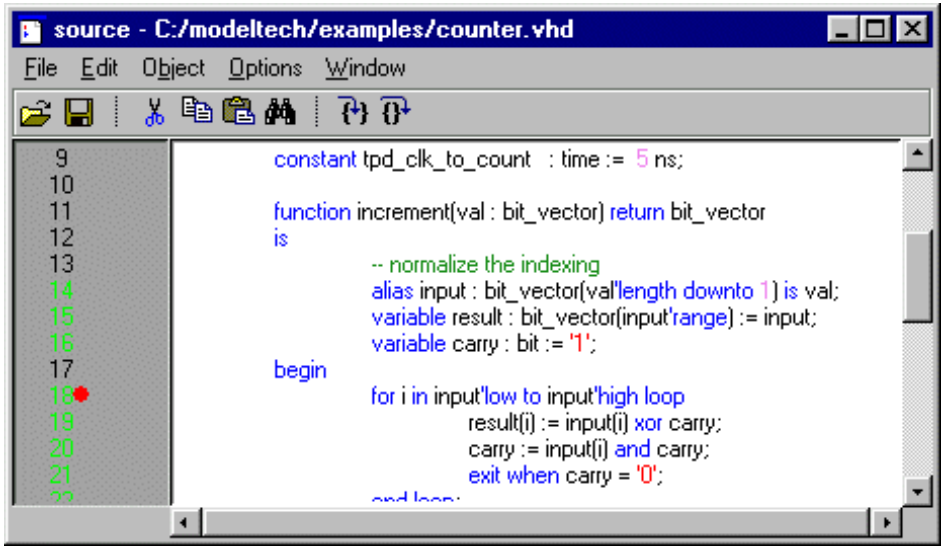
The arrow in the Source window points to the next HDL statement to be executed. (If the simulator is not evaluating a process at the time the Break occurs, no arrow will be displayed in the Source window.)

Next, you will set a breakpoint in the function on line 18.

13 Move the pointer to the VSIM Source window. Using the vertical scroll bar, scroll until line 18 is visible. Click at or near line number 18 to set the breakpoint. You should see a red dot next to the line number where the breakpoint is set. This breakpoint can be toggled on and off by clicking it.

(PROMPT: bp counter.vhd 18)

Note: Breakpoints can be set only on executable lines -- denoted by green line numbers.



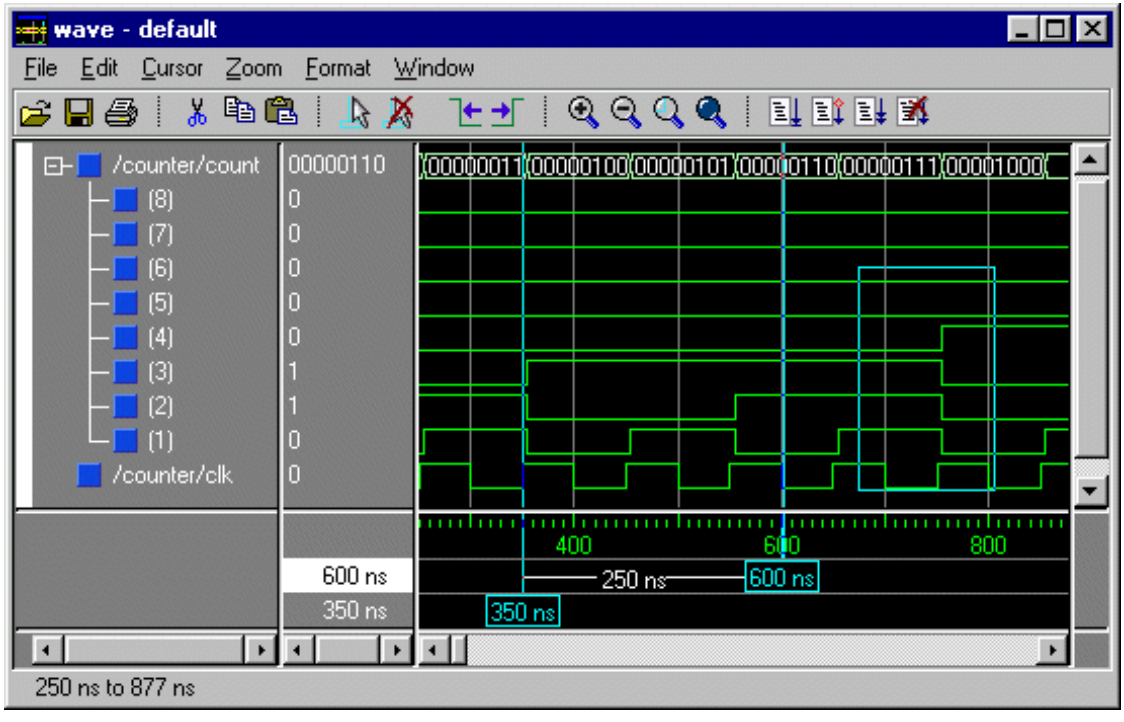
14 Select the **Continue Run** button to resume the run that you interrupted. VSIM will hit the breakpoint, as shown by an arrow in the VSIM Source window and by a Break message in the Main window. Also note that the parameters and variables within the function are displayed in the VSIM Variables window.

(PROMPT: run -continue) (MENU: Run > Continue)

15 Click the **Step** button to single-step through the simulation. Notice that the values change in the VSIM Variables window. You can keep clicking **Step** if you wish.

(PROMPT: run -step) (PROMPT: step)

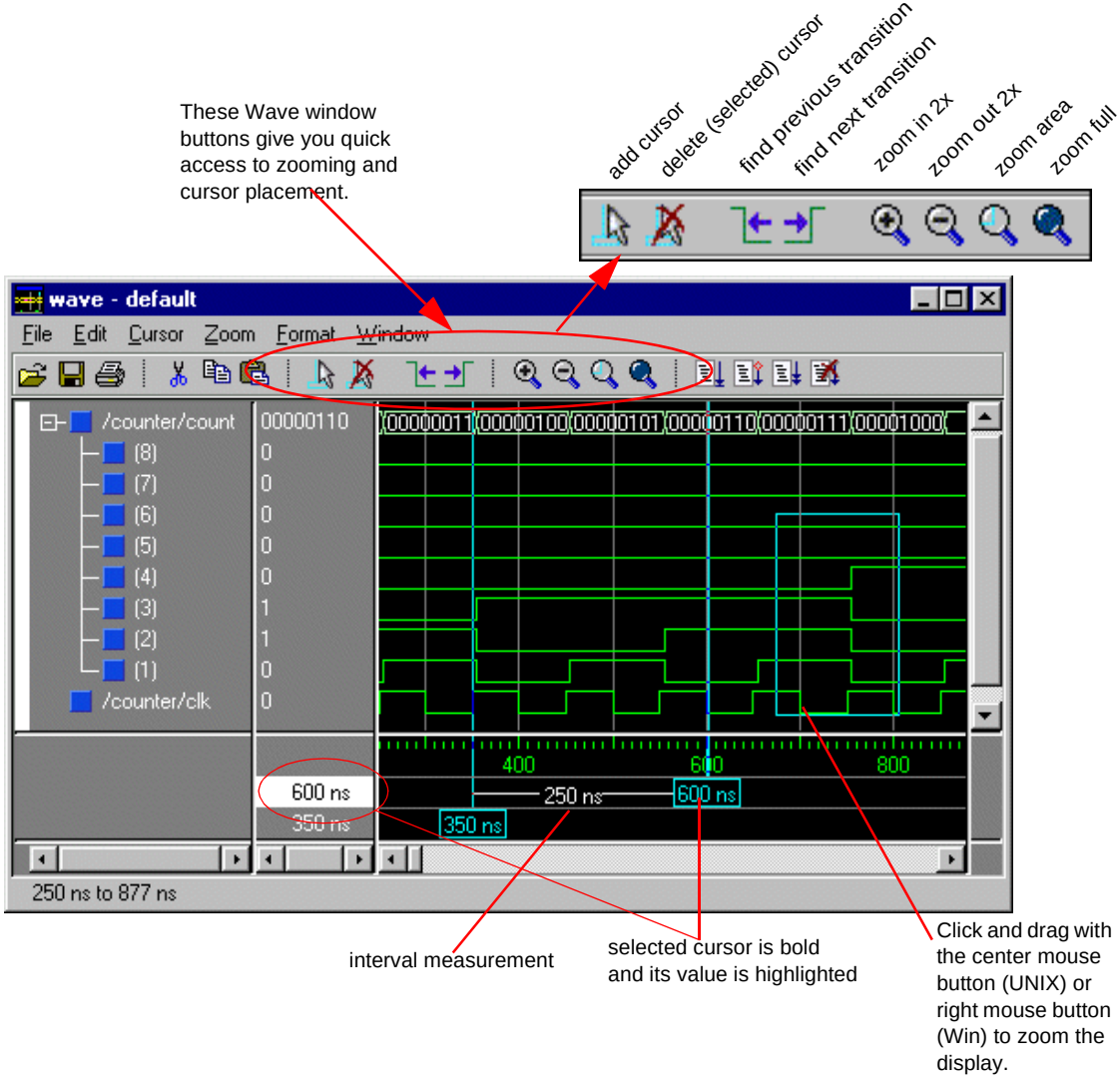
Now let's take a look at the simulation in the Wave window.



In the Wave window, you can use cursors to:

- **probe for values** - Signal values update whenever you move the cursor.
- **find signal transition times** - Click a signal edge, the cursor displays the time.
- **measure time intervals** - Time is displayed between two cursors.

16 Experiment with using the cursors, buttons, scrolling, and zooming.



17 When you're done experimenting, quit the simulator by entering the command:

```
quit -force
```

This command exits VSIM without saving data. Your window positions will be saved in the *modelsim.ini* file and the windows will close. (Refer to the *ModelSim Reference Manual* for additional information on the *modelsim.ini* file.)

Lesson 3 - Debugging a VHDL design

The goals for this lesson are:

- Show an example of a VHDL testbench - a VHDL architecture that instantiates the VHDL design units to be tested, provides simulation stimuli, and checks the results.
- Map a logical library name to an actual library.
- Change the default run length.
- Recognize assertion messages in the command window.
- Change the assertion break level
- Restart the simulation run using the **restart** command.
- Examine composite types displayed in the VSIM Variables window.
- Change the value of a variable.
- Use a strobe to trigger lines in the VSIM List window.
- Change the radix of signals displayed in the VSIM List window.

1. Return to the directory you created in Lesson 2 - Basic VHDL simulation (p21), and invoke Model*Sim*:

   for UNIX at the shell prompt:

```
vsim -gui
```

   for Windows - your option - DOS prompt, shortcut, or Start menu:

```
modelsim.exe
```

2. Enter the following command at the Model*Sim* prompt to create the a new library:

```
vlib library_2
```

3. Compile the source files into the new library by entering this command at the system prompt:

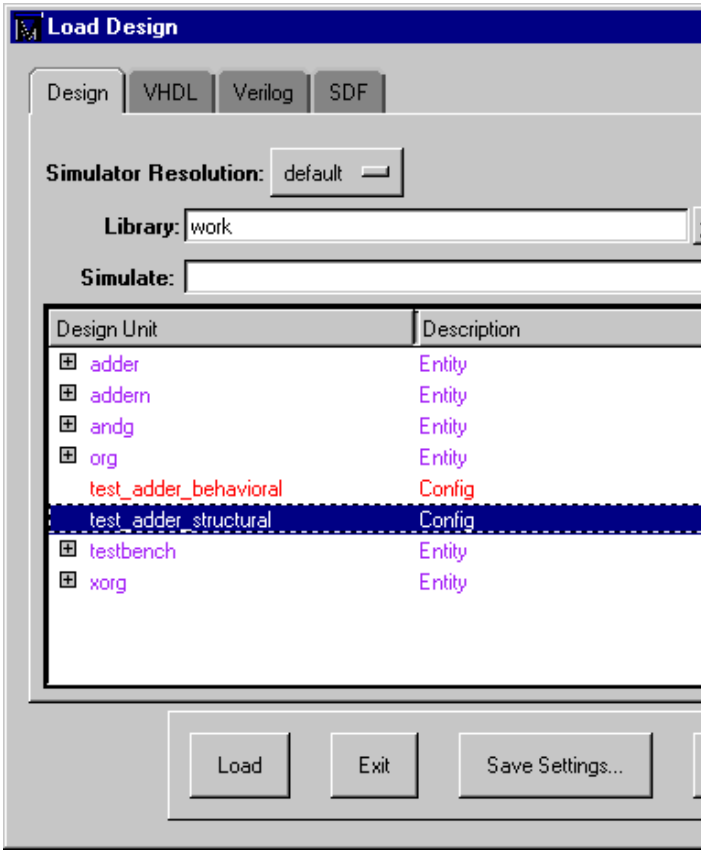
```
vcom -work library_2 gates.vhd adder.vhd testadder.vhd
```

4. Now let’s map the new library to the work library. To create a mapping you can edit the [Library] section of the *modelsim.ini* file, or you can create a logical library name with the **vmap** command:

```
vmap work library_2
```

ModelSim modifies the *modelsim.ini* file for you.

5 Start the simulator by selecting **Design > Load New Design** from the Main window, or by clicking the Load Design icon. The Load Design dialog box is displayed, as shown here.



6 Perform the following steps in this dialog box:

- Make sure that the simulator resolution is **default**. (The default resolution is **ns**.)
- Look in the Design Unit scroll box and select the configuration named **test_adder_structural**.
- Click **Load** to accept the settings.

(PROMPT: vsim -t ns work.test_adder_structural)

7 To open all of the VSIM windows, enter the following command in the Main window at the VSIM prompt:

```
view *
```

(Main MENU: View > All)

ModelSim will open all the windows in the positions you left them at the end of the last exercise if no one has run the simulator since then.

8 Drag and drop the top-level signals to the List window in the following manner: make sure the hierarchy is not expanded (no minus boxes), select all signals in the Signals window with Edit > Select All, then drag the selected signals to the List window.

(Signals MENU: View > List > Signals in Region) (PROMPT: add list *)

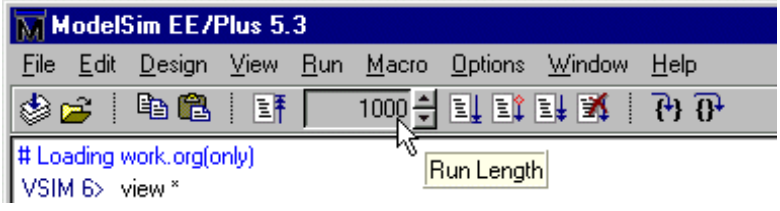
9 To add top-level signals to the Wave window, enter the command:

```
add wave *
```

(Signals MENU: View > Wave > Signals in Region) (DRAG&DROP)

10 Now change the default simulation run length to 1000 (ns) with the run length selector on the Main toolbar. Click on the field to edit the number to 1000 (notice how the arrows allow you to change the run length in increments).

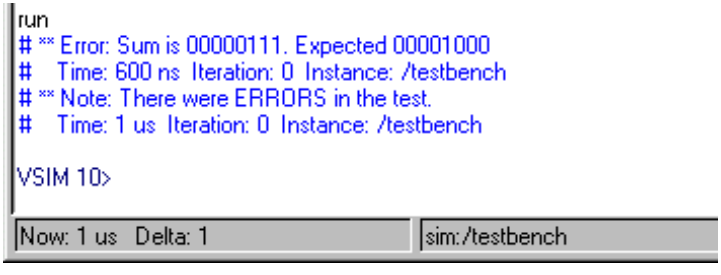
(Main MENU: Options > Simulation > Defaults)



11 Next, you will run the simulator. Select the **Run** button on the Main toolbar.

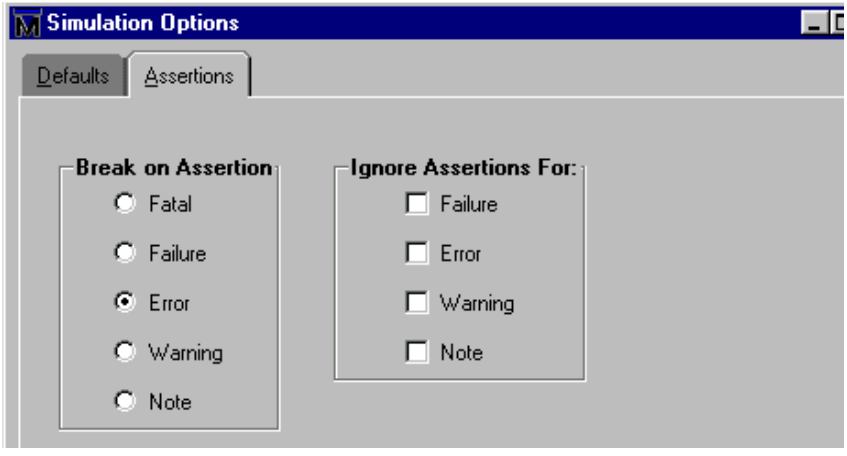
(PROMPT: run)

A message in the Main window will notify you that there was an assertion error.



Let's find out what's wrong. Perform the following steps to track down the assertion message.

12 First, change the simulation assertion options. Make this Main menu selection: **Options** > **Simulation**.

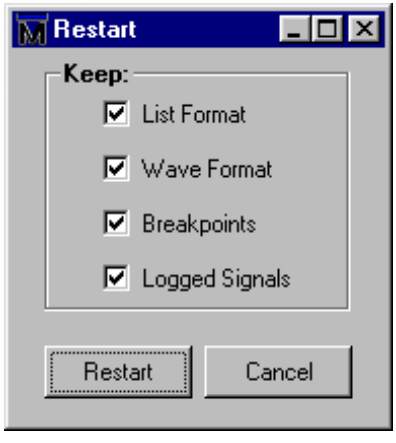


13 Select the **Assertions** page. Change the selection for **Break on Assertion** to **Error** and click **OK**. This will cause the simulator to stop at the HDL statement *after* the assertion is displayed.

14 To restart the simulation select the **Restart** button on the Main toolbar.

(Main MENU: File > Restart) (PROMPT: restart)

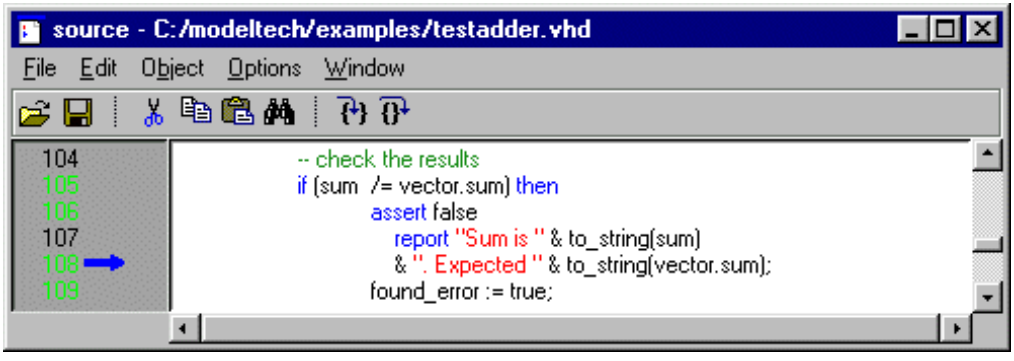
Make sure all items in the Restart dialog box are selected, then click **Restart**.



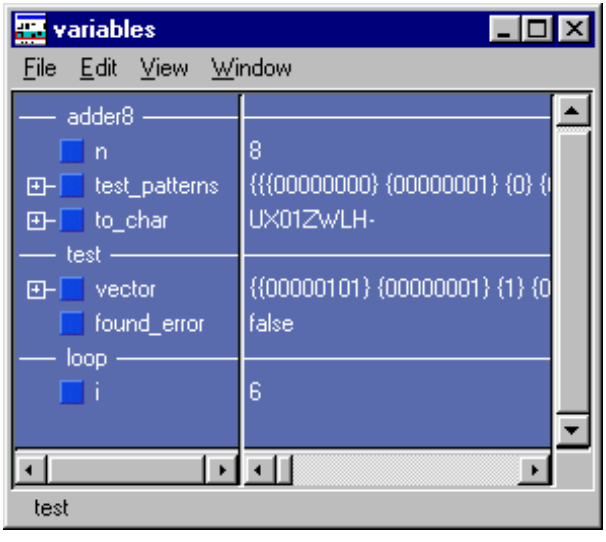
15 From the Main toolbar select the **Run** button.

(Main MENU: Run > Run 1000 ns) (PROMPT: run)

Notice that the arrow in the Source window is pointing to the statement after the assertion.

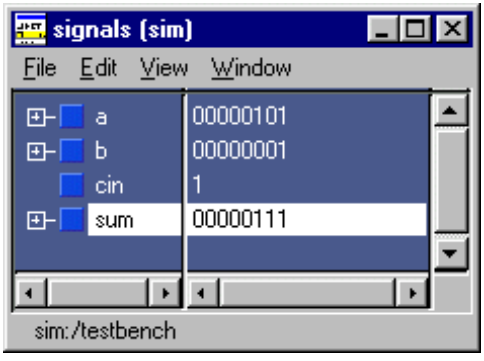


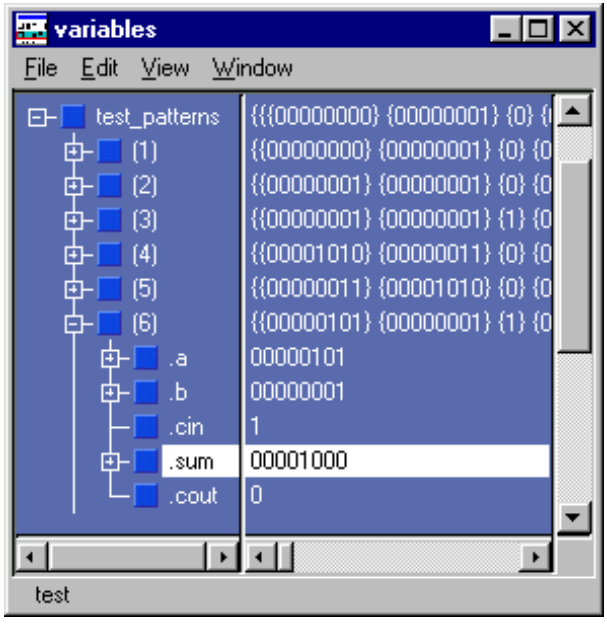
16 If you turn to the Variables window now, you can see that i = 6. This indicates that the simulation stopped in the sixth iteration of the test pattern's loop.



17 Expand the variable named **test_patterns** by clicking the [+]. (You may need to resize the window for a better view.)

18 Also expand the sixth record in the array **test_patterns(6)**, by clicking the [+]. The Variables window should be similar to the one below.





The assertion shows that the Signal **sum** does not equal the **sum** field in the Variables window. Note that the sum of the inputs **a**, **b**, and **cin** should be equal to the output **sum**. But there is an error in the test vectors. To correct this error, you need to restart the simulation and modify the initial value of the test vectors.

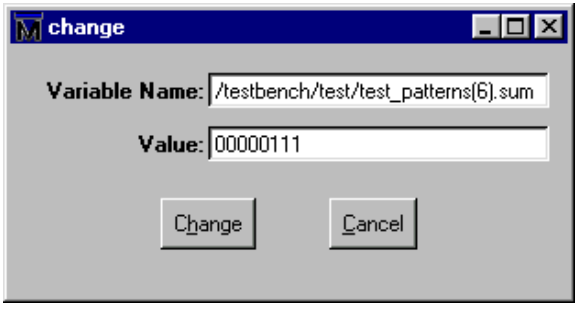
19 In the Main window, type:

```
restart -f
```

The **-f** option causes VSIM to restart without popping up the confirmation dialog.

20 Add variables to the Variables window by selecting the **test /testbench** process in the Process window.

21 In the Variables window, expand **test_patterns** and **test_pattern(6)** again. Then highlight the .**sum** record by clicking on the variable name (not the box before the name) and then use the **Edit** > **Change** menu selection.

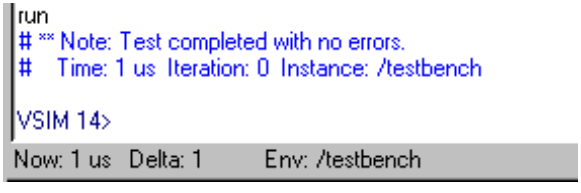


22 Select the last four bits in the value field **1000** by dragging the pointer across them. Then replace them with **0111**, and click **Change**. (Note that this is a temporary edit, you must use your text editor to permanently change the source code.)

23 Select the **Run** button from the Main toolbar.

(Main MENU: Run > Run 1000 ns) (PROMPT: run)

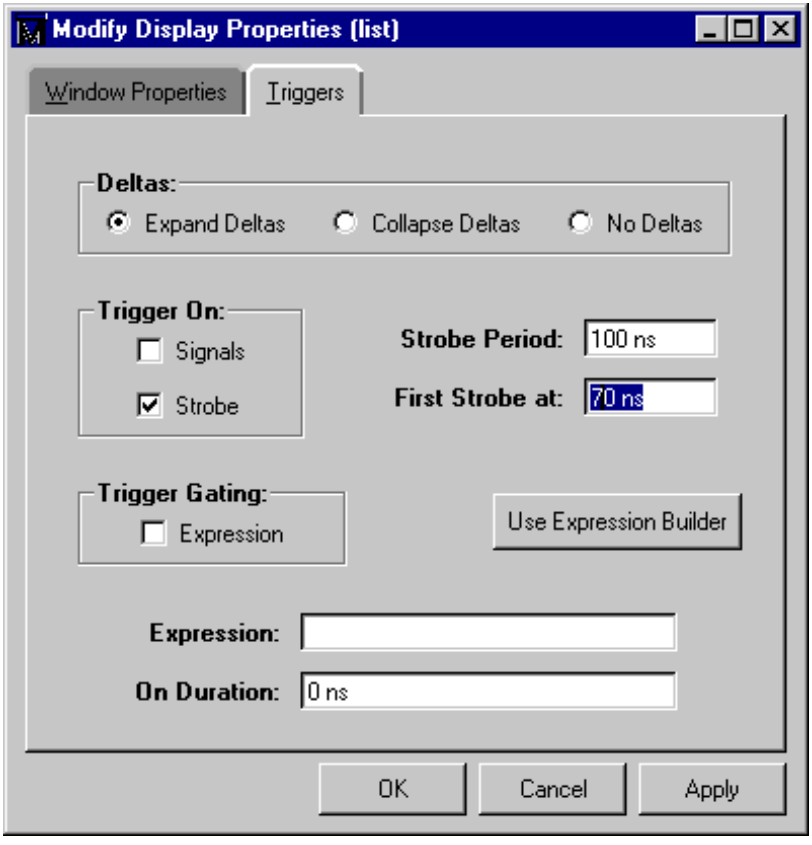
At this point, the simulation will run without errors.



Next you will learn how to change the new-line triggering for the List window.

By default, a new line is displayed in the List window for each transition of a listed signal. The following steps will change the triggering so the values are listed every 100 ns.

24 In the List window, make this menu selection: **Prop** > **Display Props**.

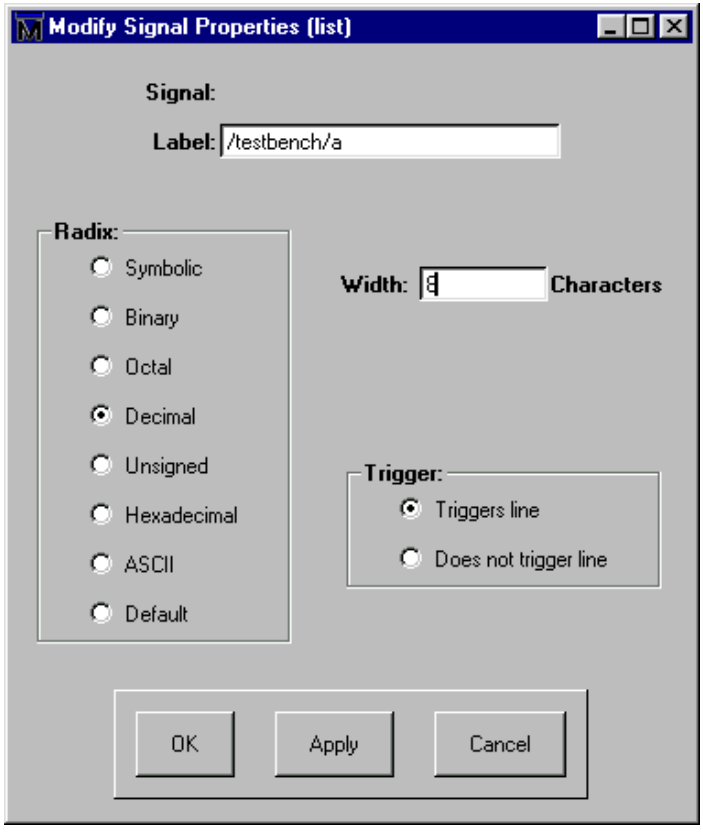


25
Perform these steps on **Triggers** page in the Modify Display Properties (list) dialog box:

- Deselect **Trigger On: Signals** to disable triggering on signals.
- Select **Trigger On: Strobe** to enable the strobe.
- Enter **100** in the **Strobe Period** field.
- Enter **70** in the **First Strobe at** field.
- Click **OK** to accept the settings.

26 Your next action will be to change the radix for a, b, and sum to decimal.

Make this List window menu selection: **Prop > Signal Props**. This will open the Modify Signal Properties (list) dialog box.



27 In the List window select the signal you want to change, then make the property changes in the dialog box. Make the following property changes:

- Select signal **a**, then click **Decimal**, then click **Apply**.
- Select signal **b**, then click **Decimal**, then **Apply**.
- Select signal **sum**, then click **Decimal**, then **OK**.

This brings you to the end of this lesson, but feel free to experiment further with the menu system. When you are ready to end the simulation session, quit VSIM without saving data by entering the following command at the VSIM prompt:

```
quit -force
```

Lesson 4 - Running a batch-mode simulation

The goals for this lesson are:

- Run a batch-mode VHDL simulation.
- Execute a macro (DO) file.
- View a saved simulation.

You'll work this lesson from a DOS or UNIX prompt.

Note: Batch-mode simulations must be run from a DOS or UNIX prompt.

1. To set up for this lesson you'll need to create a new directory and make it the current directory. Copy this file into your new directory:

```
\<install_dir>\modeltech\examples\counter.vhd
```

2. Create a new design library:

```
vlib work
```

3. Map the library:

```
vmap work work
```

4. Then compile the source file:

```
vcom counter.vhd
```

5. You will use a macro file that provides stimulus for the counter. For your convenience, a macro file has been provided with Model*Sim*. You need to copy this macro file from the installation directory to the current directory:

```
<install_dir>\modeltech\examples\stim.do
```

6. Create a batch file using an editor; name it *yourfile*. With the editor, put the following on separate lines in the file:

```
add list -decimal *
do stim.do
write format list counter.lst
```

and save to the current directory.

7 To run the batch-mode simulation, enter the following command:

```
vsim -wav saved.wav counter < yourfile
```

This is what you just did in Step 7:

- invoked the VSIM simulator on a design unit called "counter"
- the **-wav** switch instructed the simulator to save the simulation results in a log file named *saved.wav*
- used the contents of *yourfile* to specify that values are to be listed in decimal, to execute a stimulus file called *stim.do*, and to write the results to a file named *counter.lst*, the default for a design named counter

8 Since you saved the simulation results in *saved.wav*, you can view the simulation results by starting up VSIM with its **-view** switch:

```
vsim -view saved.wav
```

9 Open these windows with the VIEW menu in the Main window, or the equivalent command at the ModelSim prompt:

```
view structure signals list wave
```

Note: If you open the Process or Variables windows they will be empty. You are looking at a saved simulation, not examining one interactively; the logfile saved in *saved.wav* was used to reconstruct the current windows.

10 Now that you have the windows open, put the signals in them:

```
add wave *
add list *
```

11 Use the available VSIM windows to experiment with the saved simulation results and quit when you are ready:

```
quit -f
```

For additional information on the batch and command line modes, please refer to the *ModelSim Reference Manual*.

Lesson 5 - Executing commands at startup

The goals for this lesson are:

- Specify the design unit to be simulated on the command line.
- Edit the *modelsim.ini* file.
- Execute commands at startup with a DO file.

Start this lesson from either the UNIX or DOS prompt.

1 For this lesson, you will use a macro (DO) file that provides startup information. For convenience, a startup file has been provided with the Model*Sim* program. You need to copy this DO file from the installation directory to your current directory:

```
\<install_dir>\modeltech\examples\startup.do
```

2 Next, you will edit the system initialization file in the *\modeltech* directory to specify a command that is to be executed after the design is loaded. To do this, open the *modelsim.ini* file using a text editor and uncomment the following line in the [vsim] section of the file:

```
Startup = do startup.do
```

Then save *modelsim.ini*.

Note: The *modelsim.ini* file must be write-enabled for this change to take place. Using MS Explorer, right-click on *\<install_dir>\modeltech\modelsim.ini*, then click Properties. In the *modelsim.ini* Properties dialog box, uncheck the Read-only box and click OK.

---

3 Take a look at the DO file. It uses the predefined variable **$entity** to do different things at startup for different designs.

4 Start the simulator and specify the top-level design unit to be simulated by entering the following command at the UNIX/DOS prompt:

```
vsim counter
```

Notice that the simulator loads the design unit without displaying the Load Design dialog box. This is handy if you are simulating the same design unit over and over.

Also notice that all the windows are open. This is because the **view *** command is included in the startup macro.

5. If you plan to continue with the following practice sessions, keep Model*Sim* running. If you would like to quit the simulator, enter the following command at the VSIM prompt:

```
quit -force
```

6. You won't need the *startup.do* file for any other examples, so use your text editor to comment out the "Startup" line in *modelsim.ini*.

Lesson 6 - Tcl/Tk and ModelSim

This lesson is divided into several Tcl examples intended to give you a sense of Tcl/Tk's function within Model*Sim*. The examples include a custom simulation interface created with Tcl/Tk. You must be using Model*Sim* /PLUS or Model*Sim* /VHDL to complete these exercises.

The goals for this lesson are:

Example 1 - Create a "hello world" button widget. (p52)

Example 2 - Add a procedure that gets called by a button push. (p52)

Example 3 - Simulate an intersection with traffic lights. (p53)

Example 4 - Draw a state machine that represents the simulation. (p57)

More information on Tcl/Tk

Tcl print references

Sources of information about Tcl include *Tcl and the Tk Toolkit* by John K. Ousterhout, published by Addison-Wesley Publishing Company, Inc., and *Practical Programming in Tcl and Tk by* Brent Welch published by Prentice Hall.

Tcl online references

The following are a few of the many Tcl references available:

- When using Model*Sim* make this VSIM Main menu selection: **Help > Tcl Man Pages**.
- Tcl man pages are also available at: www.elf.org/tcltk-man-html/contents.htm
- Tcl/Tk general information is available from the Tcl/Tk Consortium: www.tclconsortium.org
- The Scriptics Corporation, John Ousterhout's company (the original Tcl developer): www.scriptics.com.

How Tcl/Tk works with Model*Sim*

Model*Sim* incorporates Tcl as an embedded library package. The Tcl library consists of a parser for the Tcl language, routines to implement the Tcl built-in commands, and procedures that allow Tcl to be extended with additional commands specific to Model*Sim*.

Model*Sim* generates Tcl commands and passes them to the Tcl parser for execution. Commands may be generated by reading characters from an input source, or by associating command strings with Model*Sim*'s user interface features, such as menu entries, buttons, or keystrokes.

When the Tcl library receives commands it parses them into component fields and executes built-in commands directly. For commands implemented by Model*Sim*, Tcl calls back to the application to execute the commands. In many cases commands will invoke recursive invocations of the Tcl interpreter by passing in additional strings to execute (procedures, looping commands, and conditional commands all work in this way).

Model*Sim* gains a programming advantage by using Tcl for its command language. Model*Sim* can focus on simulation-specific commands, while Tcl provides many utility commands, graphic interface features, and a general programming interface for building up complex command procedures.

By using Tcl, Model*Sim* need not re-implement these features, a benefit that allows it's graphic interface to remain consistent on all platforms. (The only vestige of the host platform's graphic interface is the window frame manager.)

The custom-traffic-light interface

The subject of our main Tcl/Tk lesson is a simple traffic-light controller. The system is comprised of three primary components: a state machine, a pair of traffic lights, and a pair of traffic sensors. The components are described in three VHDL files: traffic.vhd (the state machine), queue.vhd (the traffic arrival queue) and tb_traffic.vhd (the testbench).

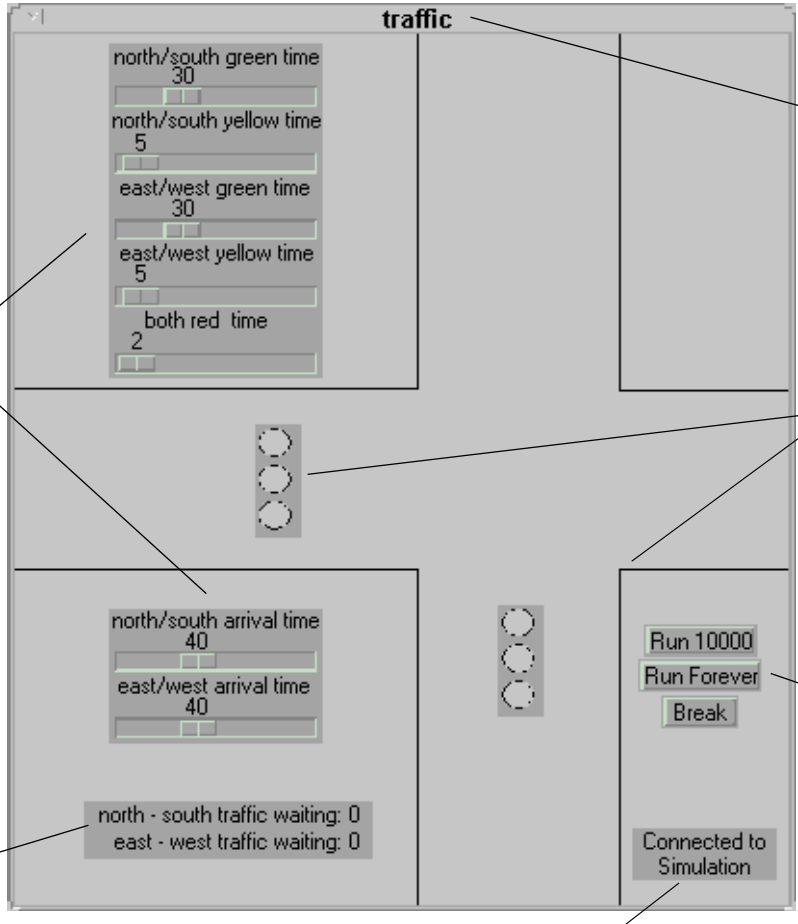
You could, of course, simulate this system with Model*Sim*'s familiar interface, but Tcl/Tk provides us the option to try something different. Since we're simulating something most of us have seen and experienced before, we can create an intuitive interface unique to the simulation.

Assumptions

The example assumes that source files were previously compiled. Here's how it works:

VHDL source files describe the system	**Tcl procedures create and connect the interface, plus the source files, to Model*Sim***	**Model*Sim* commands are run via the new interface using the Tcl procedures**
	draw_intersection	
traffic.vhd queue.vhd tb_traffic.vhd	connect_lights	vsim -lib vhdl/work tb_traffic examine -value <light_timing>
	draw_queues	
	draw_controls	force -freeze $var $val ns

The result is a traffic intersection interface similar to this illustration:



wm widget

Calls to the operating system window manager to create the "traffic" window.

frame and scale widget

A scale widget within a frame widget creates an analog entry device for a minimum to a maximum value and invokes the VSIM force command

canvas widget

The background, lines and traffic lights are created with the canvas widget.

button widgets

Each button invokes the indicated VSIM run or break command.

entry widget

The entry widget (contained within a frame widget) can facilitate entry, or in this case, provides a display for a VSIM examine command.

label widget

A static label is added to the end of the connect_lights procedure to indicate connection to the simulator.

Tk widgets

The intersection illustration points out several Tcl/Tk "widgets". A widget is simply a user interface element, like a menu or scrolled list. Tk widgets are referenced within Tcl procedures to create graphic interface objects. The Tk tool box comes with several widgets, additional widgets can be created using these as a base.

Controlling the simulation

The components of the intersection interface have the following effect within Model*Sim*:

Intersection control used	**Effect in Model*Sim***
Run 1000 button	invokes the run command for 1000 ns
Run Forever button	invokes the run -all command
Break button	invokes the break command
light timing control	invokes the force command with the arguments for the indicated signal and time
arrival time control	invokes the force command with the arguments for the indicated direction and time
waiting queue	any time you change a control the examine command is invoked to display the value of the waiting queue

Saving time

Since several intersection controls invoke a VSIM command and arguments with a single action (such as the movement of a slider), this custom interface saves time compared to invoking the commands from the command line or Model*Sim* menus.

Copies of the original example files

Additional copies of the Tcl example files from these exercises are located in the *..\examples\tcl_tutorial\originals* directory.

Solutions to the examples

Throughout the traffic intersection examples you will be modifying Tcl files to complete the final intersection. You will find a completed set of intersection

examples ready-to-run in the *tcl_tutorial\solutions* directory. Invoke these commands from the Model*Sim* prompt to run the intersection:

```
cd solutions
do traffic.do
```

Viewing files

If you would like to view the source for any of the Tcl files in our examples, use the **notepad** command at either the Model*Sim* or VSIM prompt.

```
notepad <filename>
```

Most files are opened in read-only mode by default; you can edit the file by deselecting **read only** from the notepad **Edit** menu.

The Tcl source command

The Tcl **source** command reads the Tcl file into the Tcl interpreter, which parses the procedures for use within the current environment. Once sourced, a Tcl procedure can be called from the Model*Sim* prompt as shown in the syntax below. VSIM executes the instructions within the procedure.

Syntax

```
source <tcl filename>
<tcl procedure name>
```

Arguments

`<tcl filename>`
The Tcl file read into the VSIM Tcl interpreter with the **source** command.

`<tcl procedure name>`
The Tcl procedure defined within <tcl filename>, called from the Model*Sim* prompt, and executed by VSIM.

The *traffic.do* file is a good example of the **source** command syntax (the file is a macro that runs the traffic light simulation). Check it out with the notepad:

```
notepad traffic.do
```

Example shortcuts

To save some typing, copy the commands from the PDF version of these instructions and paste them at the Model*Sim* prompt. Paste with the right (2 button

mouse), or middle (3 button mouse). You can also select a Model*Sim* or VSIM prompt from the Main transcript to paste a previous command to the current command line.

Make a transcript DO file

You can rerun the commands executed during the current session with a Do file created from the Main transcript. Make the DO file by saving the transcript with the **File > Save Main As** menu selection at any time during the exercises. Run the DO file to repeat the commands (do <do filename>).

Preparing for the Tcl/Tk examples

These steps must be completed before running the Tcl examples.

1. Create, and change to a new working directory for the Tcl/Tk exercises. Copy the lesson files in the following directory (include all subdirectories and files) to your new directory:

```
<install_dir>\modeltech\examples\tcl_tutorial
```

2. Make the new directory the current directory, then invoke Model*Sim*:

   for UNIX

```
vsim -gui
```

   for Windows (from a shortcut or Start > Run, etc.)

```
modelsim.exe
```

3. At the ModelSim prompt, create a **work** library in the */vhdl* directory:

```
vlib vhdl/work
```

4. Map the **work** library.

```
vmap work vhdl/work
```

5. Compile the VHDL example files with these commands (or the Compile dialog box):

```
vcom vhdl/traffic.vhd
vcom vhdl/queue.vhd
vcom vhdl/tb_traffic.vhd
```

Now you're now ready to run the examples.

Example 1 - Create a "hello world" button widget.

Before you begin the examples make sure you have completed "Preparing for the Tcl/Tk examples" (p51).

In this example you will study a "hello world" button that prints a message when pressed.

1. Source the Tcl file from the Model*Sim* prompt:

```
source hello.tcl
```

then run the procedure defined within *hello.tcl*:

```
hello_example
```

The file *hello.tcl* was read into the VSIM Tcl interpreter. The instructions in *hello_example* procedure were then executed by VSIM, and "Hello World" was printed to the Main transcript. Selecting the button will print the message again.

You've just created your first top-level widget!

2. Invoke the *hello_example* procedure again and notice how the new button replaces the original button. The procedure destroyed the first button and created the new one. Get a closer look at the source Tcl file with the **notepad**:

```
notepad hello.tcl
```

Close the hello_example window when you're done.

Example 2 - Add a procedure that gets called by a button push.

Before you begin this example make sure you have completed "Preparing for the Tcl/Tk examples" (p51).

In this example you will study a larger Tcl example, and add a procedure that gets called by a button push.

This example will display all of the gif images in the images directory. Each button has a binding attached to it for "enter" events, and a binding for a mouse button press. When the mouse enters the button graphic, the image file name is printed to the Main window. When the mouse button is pushed its "widget" name will be printed to the Main window.

1. Build an image viewer by invoking this command, and calling this procedure:

```
source images.tcl
image_example
```

2. Drag the mouse across the buttons and notice what happens in the Main transcript.

   Push one of the buttons; you will see an error dialog box. You can solve this problem by modifying the *images.tcl* file.

3. To view the source file press the **See Source Code** button at the bottom of the image display or invoke **notepad** at the Model*Sim* prompt:

```
notepad images.tcl
```

   You'll find that the *pushme* procedure is missing; it's commented out in *images.tcl*.

4. Search for "proc push" using the **Edit > Find** menu selection in the notepad.

   Remove the comments (the "#" symbols) to return the function to your source, then close the image window with the **Destroy** button.

5. Once the *pushme* procedure is in place it will print its one parameter, the object name, to the transcript.

   After you have added the *pushme* procedure to your source, you need to resource and rerun the Tcl procedure with these commands (use the up arrow to scroll through the commands or do !source):

```
source images.tcl
image_example
```

   Press all the buttons and notice the object names in the Main transcript. Close the image example window when you're done.

Example 3 - Simulate an intersection with traffic lights.

In this example you'll simulate an intersection with traffic lights. The simulation interface you create allows you to run "what if" scenarios efficiently.

Introduction of the traffic intersection widget

This portion of our example introduces the traffic intersection widget. You'll add other widgets to the intersection to create a custom traffic simulation environment.

Once again, make sure you have completed "Preparing for the Tcl/Tk examples" (p51) before working this example.

1. Draw the intersection by invoking this command and procedure at the Model*Sim* prompt:

```
source intersection.tcl
draw_intersection
```

2 From the Model*Sim* prompt, use the procedure set_light_state to change the color of the lights:

```
set_light_state green .traffic.i.ns_light

set_light_state green .traffic.i.ew_light
```

You can use the Copy and Paste buttons on the Main toolbar to help build instructions from previous commands.

3 View the source code with this command at the Model*Sim* prompt:

```
notepad intersection.tcl
```

You can locate the *set_light_state* procedure with **Edit > Find** from the Main menu (it's located toward the middle of the file).

Connect traffic lights to the simulation

Using the intersection widget, you will add *when* statements to connect the lights to the real simulation. Once the connection is made you will simulate the traffic light controller and watch the lights change.

We'll use VSIM *when* statements to condition the simulation to call our Tcl program when a desired simulation condition happens.

For our example, the desired condition is the state of the lights. Whenever the state of the light in the simulation changes, we want to change the color of the light on the screen.

4 Load the VHDL libraries you compiled in preparation for these examples using this command at the Model*Sim* prompt:

```
vsim tb_traffic
```

Be sure you invoke this command before the start of the connect_lights procedure, if you don't load the libraries, you won't have a design to simulate.

5 Connect the lights to the simulation with this command and procedure:

```
source lights.tcl
connect_lights
```

Try running the simulation now; select either run button on the intersection. Select **Break** if you used the **Run Forever** button. Notice how the Source window opens

and indicates the next line to be executed. (If the simulator is not evaluating an executable process when the break occurs, the Source window will not open.) Only the East/West lights are working. You can make both lights work by editing the *lights.tcl* file.

6 Edit *lights.tcl* with the **notepad** to add a *when* statement for the North/South light.

```
notepad lights.tcl
```

You need to add this because the current statement is for the East/West light only. You'll find the solution commented. (Remember to change the read-only status of the file so you can edit it.)

You'll find the code commented-out toward the end of the file (try Edit >Find again).

7 After you have made the changes, reload and run the simulation again.

```
source lights.tcl
connect_lights
```

Both lights are now working.

Note: Remember, if you need to return to the original Tcl files (maybe you've edited the file and it doesn't work right) you'll find the files in the *tcl_tutorial/originals* directory.

Add widgets to display simulation information

Running the lights may be interesting, but not very useful - let's add some displays that will tell us what's happening to the cars at the intersection.

Now you will add queue widgets to display the sum of the length of each pair of queues as we simulate.

8 The East/West widget for displaying the total East/West queue length is already provided. Let's edit the source to add a display for the North/South direction. Use the **notepad**:

```
notepad queues.tcl
```

The solution is commented out in *queues.tcl*.

The Queue Display widget consists of an enclosing frame with two label widgets. The first label is a simple text string. The second label is the value of the queue length. The text in the second label will be updated whenever the queue lengths change.

The update is accomplished by using a **when** statement.

9 After you have added your North/South widget, run your program by invoking this command:

```
source queues.tcl
draw_queues
```

According to the traffic indicators, the cars are leaving the intersection at the same rate. That seems fair, but if you are designing an intersection that responds to the traffic flow into the intersection you might want to change the light cycles. Perhaps one of the directions has more incoming traffic than the other.

Adding controls, in the form of scale widgets, allows you to quickly change the assumptions about traffic flow into the intersection.

Add "scale" widgets to control the simulation

Next you will add Tk "scale" widgets that will control the arrival rates and the length of the lights.

10 The East/West widget for controlling the East/West queue inter-arrival time is provided. You'll edit the source code to add controls for the North/South direction. Use this command:

```
notepad controls.tcl
```

You can remove the comments in the code to make this change.

Similarly, add the North/South widget for controlling the length of the lights. The East/West widget for light control is provided. (You can remove the comments in the code to make this change as well.)

These control widgets are implemented using the Tk "scale" widgets, enclosed in a frame.

When the value of the scale widget changes, it calls the command specified with the **-command** option on each scale.

11 After you have added your North/South widgets, run your program with this command:

```
source controls.tcl
draw_controls
```

Now you have a complete intersection interface. Try the run buttons and the slider scales. You can also view the simulation with Model*Sim*'s GUI. Check the Source window to view the VHDL files, and add signals to a Wave window

(**add wave ***). You can also change the run length in the Main window. Try using the Run buttons in the Main window and the intersection window.

Keep the intersection simulation running to complete the next example. If you want to recreate the final intersection environment quickly, invoke these commands from the ModelSim prompt (after "Preparing for the Tcl/Tk examples" (p51)).

```
cd solutions
do traffic.do
```

Example 4 - Draw a state machine that represents the simulation.

In this final example you will draw a state machine representing the simulation, and connect it to the state signal inside the traffic light controller. Each transition that the controller makes is displayed as it happens.

The intersection environment from the previous example needs to be running for this example. To get it running quickly, invoke these commands from the Model*Sim* prompt (after "Preparing for the Tcl/Tk examples" (p51)).

```
cd solutions
do traffic.do
```

1. Run the state machine with these commands:

```
source state-machine.tcl
draw_state_machine
```

Let’s make some changes to the light colors and transition arrows.

2. Open the source file with this command:

```
notepad state-machine.tcl
```

Note the "ModelSim EXAMPLE part 1" comments in the file. You can change "both_red" state coordinates from x = 125 and y = 50 to any coordinates. (You may need to uncheck the **read only** selection in the Edit menu before making changes.)

3. Note the "ModelSim EXAMPLE part 2" comments in the file. You can change the transition arrow coordinates to correspond with the new "both_red" state coordinates.

4. Note the "ModelSim EXAMPLE part 3" comments in the file. Change the active color from "black" to "purple".

5 Reuse the original commands when you're ready to run the state machine (remember, to copy a previous command to the current command line, select the previous Model*Sim* prompt):

```
source state-machine.tcl
draw_state_machine
```

Notice the changes. Try some additional changes if you wish.

This is the end of the Tcl/Tk examples. Continue to modify and test the examples if you wish; you can recover the original files at any time in the *tcl_tutorial\originals* directory.

Lesson 7 - Basic Verilog simulation

You must be using Model*Sim*/PLUS or Model*Sim*/VLOG for this lesson.

The goals for this lesson are:

- Compile a Verilog design.
- Examine the hierarchy of the design.
- List signals in the design.
- Change list attributes.
- Set a breakpoint.
- Add and remove cursors in the waveform display.

If you've completed any previous VHDL lesson you'll notice that the Verilog and VHDL simulation processes are almost identical.

1. Create and change to a new directory to make it the current directory.

2. Copy the Verilog files (files with ".v" extension) from the *\<install_dir\>\modeltech\examples* directory into the current directory.

   Before you can compile a Verilog design, you need to create a design library in the new directory. If you are only familiar with interpreted Verilog simulators such as Cadence Verilog XL this will be a new idea for you. Since Model*Sim* is a compiled Verilog, it requires a target design library for the compilation. Model*Sim* can compile both VHDL and Verilog code into the same library if desired.

3. Invoke ModelSim:

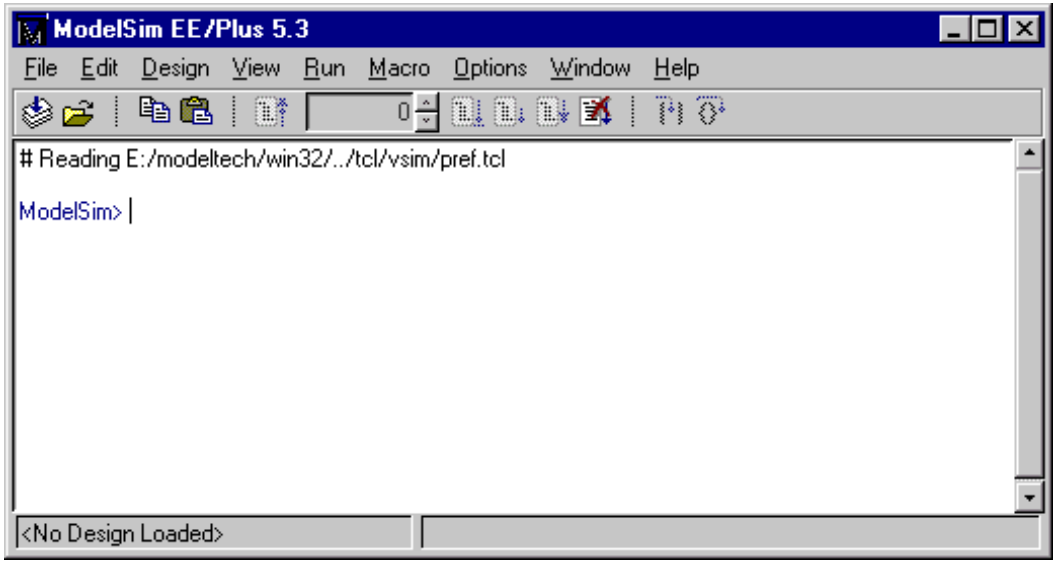
   for UNIX at the shell prompt:

```
vsim -gui
```

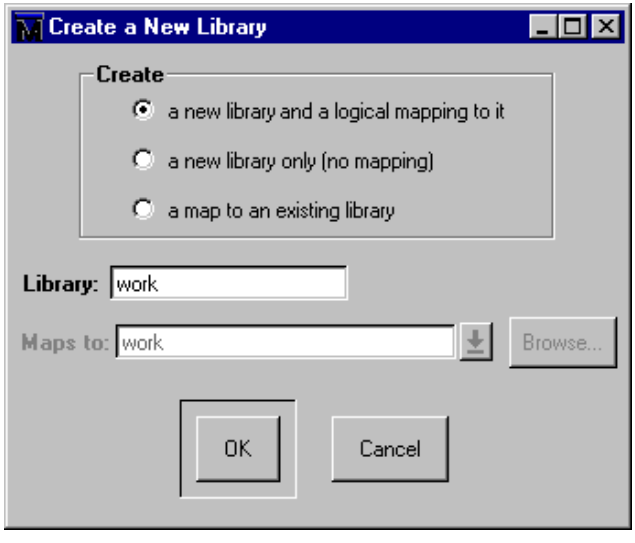
   for Windows - your option - DOS prompt, shortcut, or Start menu:

```
modelsim.exe
```

   This opens the Model*Sim* Main window without loading a design.



4 Before you compile a source file, you'll need a design library to hold the compilation results. To create a new design library, make this menu selection in the Main window: **Design > Create a New Library**. (PROMPT: vlib work)



In the Create a New Library dialog box, select **Create: a new library and a logical mapping to it**. Make sure **Library:** indicates **work**, then select **OK**. This creates a subdirectory named *work* - your design library - within the current directory. This subdirectory contains a special file named *_info*. (Do not create these using UNIX or Windows commands—always use the Library menu or the **vlib** command from either the Model*Sim* or UNIX/DOS prompt.)

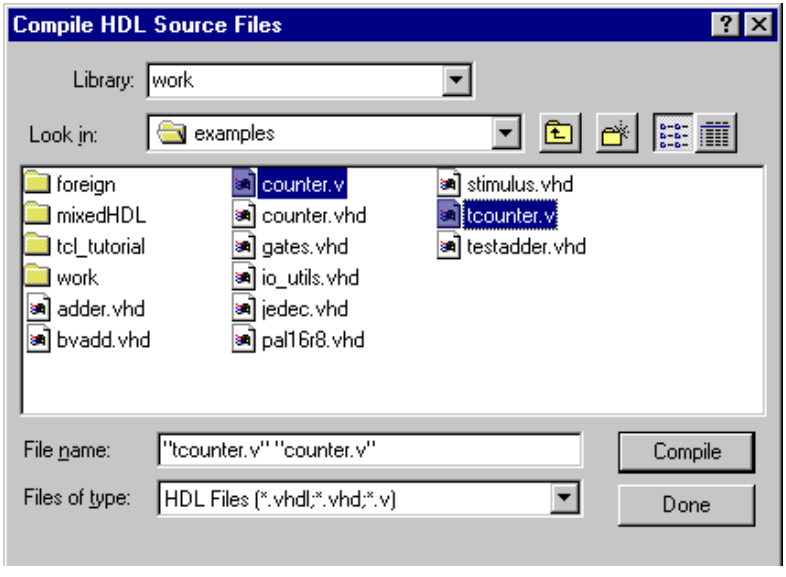
Note: Remember, a library directory should not be created using UNIX/DOS commands - always use the Main Design menu or the **vlib** command.

Next, you'll compile the Verilog design.

The example design consists of two Verilog source files, each containing a unique module. The file *counter.v* contains a module called **counter**, which implements a simple 8-bit binary up-counter. The other file, *tcounter.v*, is a testbench module (**test_counter**) used to verify **counter**. Under simulation you will see that these two files are configured hierarchically with a single instance (instance name **dut**) of module **counter** instantiated by the testbench. You'll get a chance to look at the structure of this code later. For now, you need to compile both files into the **work** design library.

5 Compile the *counter.v*, and *tcounter.v* files into the **work** library by selecting the **Compile** button on the toolbar:

(PROMPT: vlog counter.v)



This opens the Compile HDL Source Files dialog box.

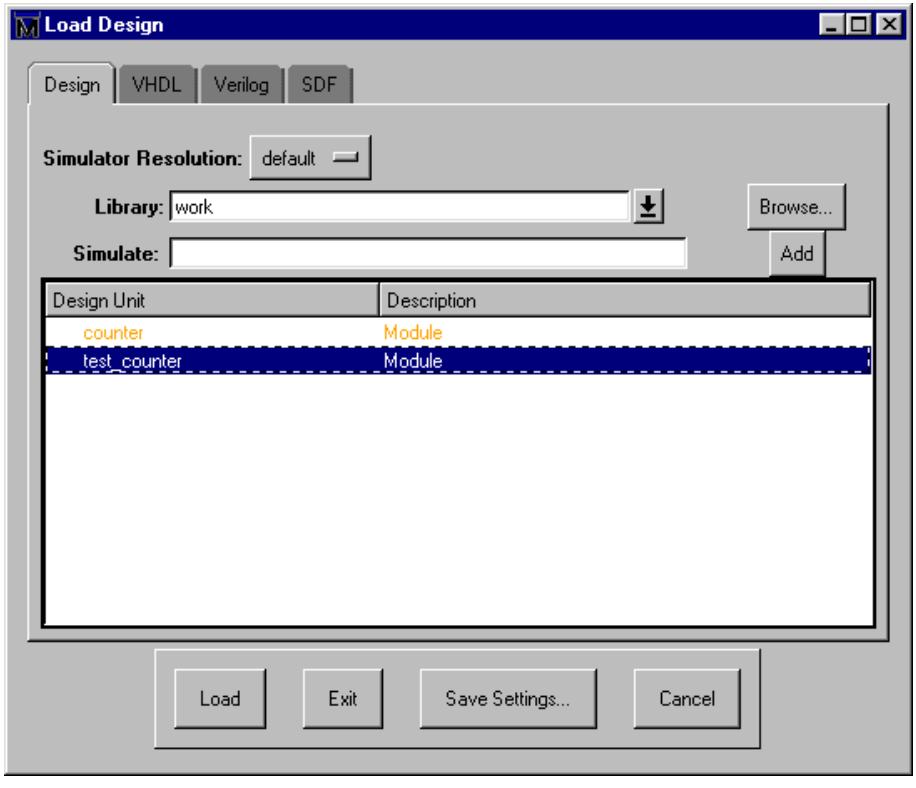
Complete the compilation by selecting both files. **Control+click** (left mouse button) on *counter.v*, then *tcounter.v* from the file list and choose **Compile**, then **Done**.

Note: The order in which you compile the two Verilog modules is not important (other than the source-code dependencies created by compiler directives). This may again seem strange to Verilog XL users who understand the possible problems of interface checking between design units, or compiler directive inheritance. Model*Sim* defers such checks until the design is loaded by VSIM (the HDL simulator). So it doesn't matter here if you choose to compile *counter.v* before or after *tcounter.v*.

6 Start the simulator by selecting the **Load Design** button from the toolbar:

(PROMPT: vsim test_counter)

The Load Design dialog box comes up, as shown below.



The Load Design dialog box allows you to select a design unit to simulate from the specified library. You can also select the resolution limit for the simulation. The default library is **work** and the default resolution is 1 ns.

7 **Design Unit: test_counter** and click **Load** to accept these settings.

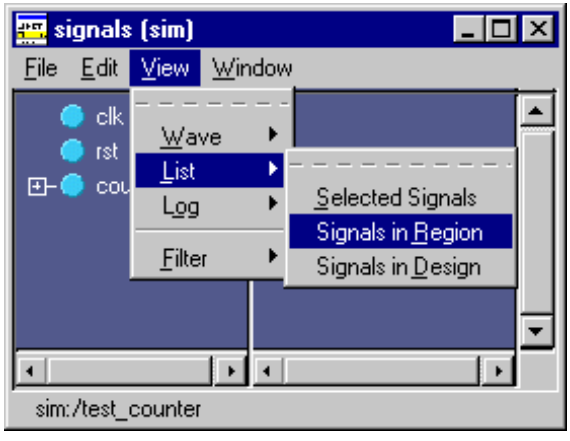
8 Bring up the Signals, List and Wave windows by entering the following line at the VSIM prompt within the Main window:

```
view signals list wave
```

(Main MENU: View > <window name>)

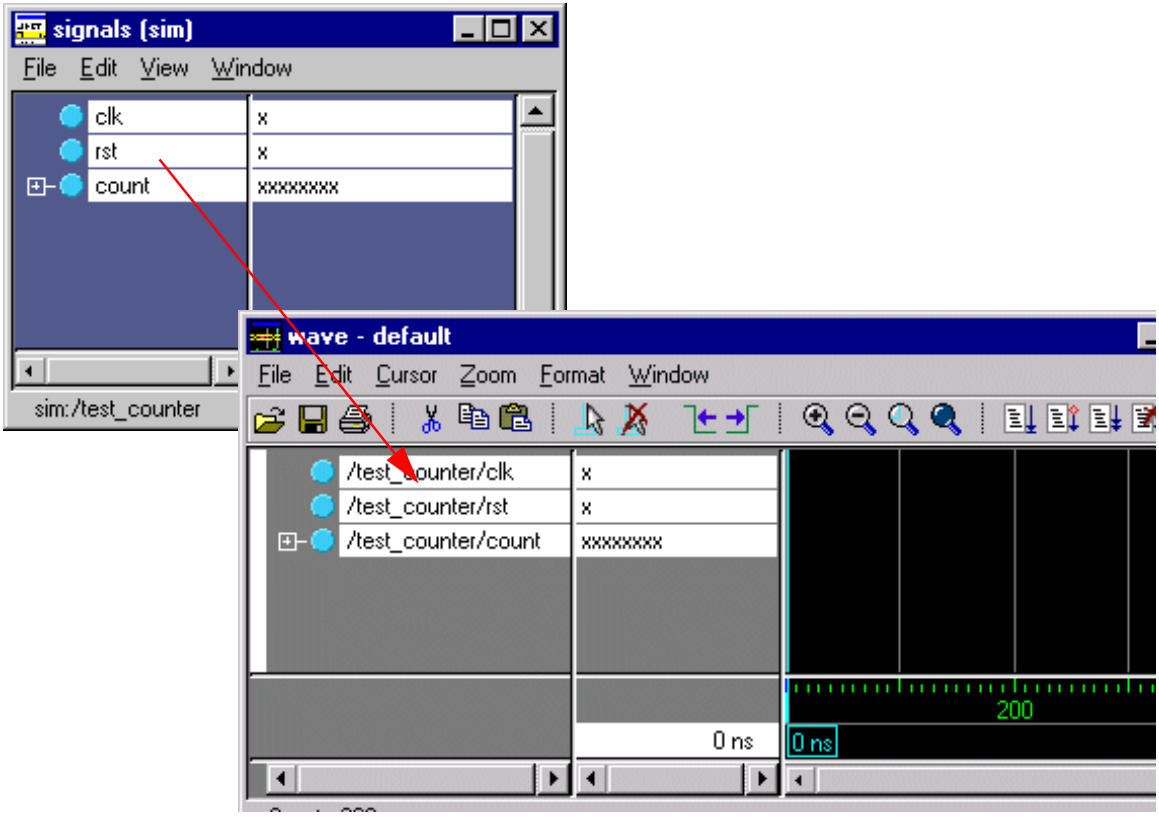
9 To list the top-level signals, move the pointer to the Signals window and make this View menu selection: **View > List > Signals in Region**.

(PROMPT: add list \counter*)



10 Now let's add signals to the Wave window with Model*Sim*'s drag and drop feature.

In the Signals window, **control-click** on each of the *clk*, *rst*, and *count* signals to make a group selection. Click and hold on the group one more time, then drag it to either the pathname or the values pane of the Wave window.

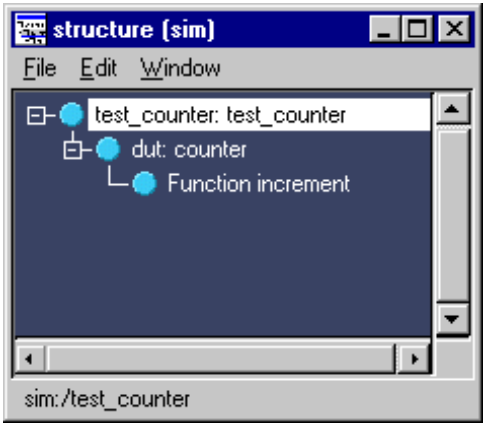


HDL items can also be copied from one window to another (or within the Wave and List windows) with the Edit > Copy and Edit > Paste menu selections. You can also delete selected items with the Edit > Delete selection.

11 Next open the Structure and Source windows. From the **Main window** make these menu selections: **View > Structure** and **View > Source**.

(PROMPT: view structure source)

12 Rearrange the windows to give yourself a clear view of all open windows (try the **Window > Initial Layout** menu selection), then click inside the Structure window.



The Structure window shows the hierarchical structure of the design. By default, only the top level of the hierarchy is expanded. You can navigate within the hierarchy by clicking on any line with a "+" (expand) or "-" (contract) symbol. The same navigation technique works anywhere you find these symbols within Model*Sim*.

By clicking the "+" next to **dut: counter** (as shown here) you can see all three hierarchical levels: **test_counter**, **counter** and a function called **increment**. (If **test_counter** is not displayed you simulated **counter** instead of **test_counter**.)

13 Click on **Function increment** and notice how other VSIM windows are automatically updated as appropriate.

Specifically, the Source window displays the Verilog code at the hierarchical level you selected in the Structure window. The source-file name is also displayed in the Source window title bar.

Using the Structure window in this way is analogous to scoping commands in interpreted Verilogs.

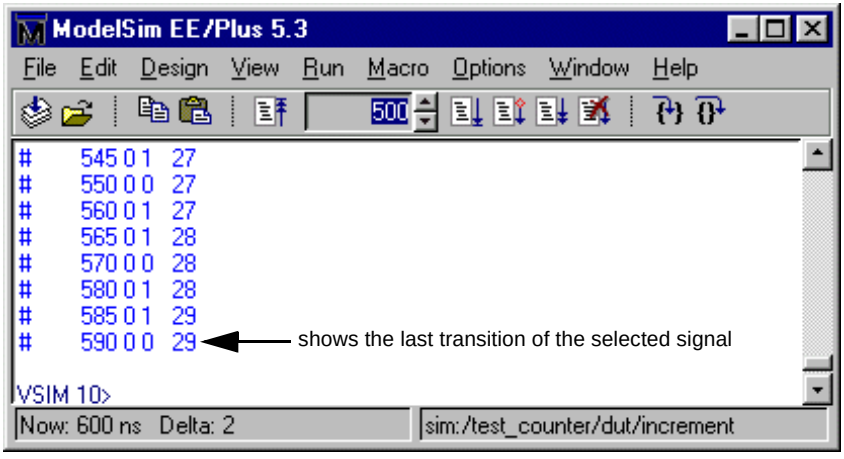
For now, make sure the **test_counter** module is showing in the Source window by clicking on the top line in the Structure window.

14 Now you will exercise different Run functions from the toolbar.

15 Select **Run** button on the Main toolbar. This causes the simulation to run and then stop after 100 ns (the default simulation length).

(PROMPT: run) (MENU: Run > Run 100 ns)

16 Next change the run length to 500 on the **Run Length** selector and select the **Run** button again.



Now the simulation has run for a total of 600ns (the default 100ns plus the 500 you just asked for). A status bar reflects this information at the bottom of the Main window.

17 The last command you executed (**run 500**) caused the simulation to advance for 500ns. You can also advance simulation to a specific time. Type:

```
run @ 3000
```

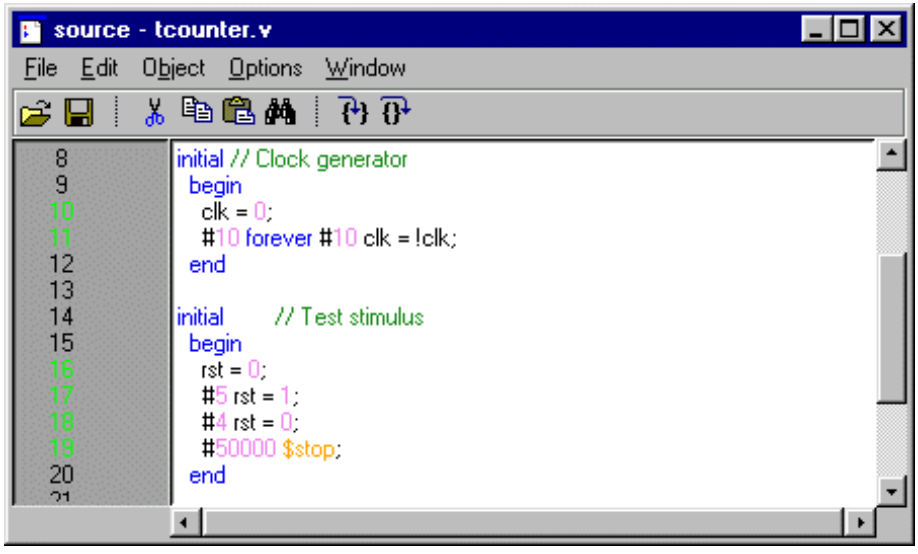
This advances the simulation to time 3000ns. Note that the simulation actually ran for 2400ns (3000 - 600).

18 Now select the **Run All** button from the Main toolbar. This causes the simulator to run forever.

(PROMPT: run -all) (Main MENU: Run > Run -All)

19 Select the **Break** button to stop the run.

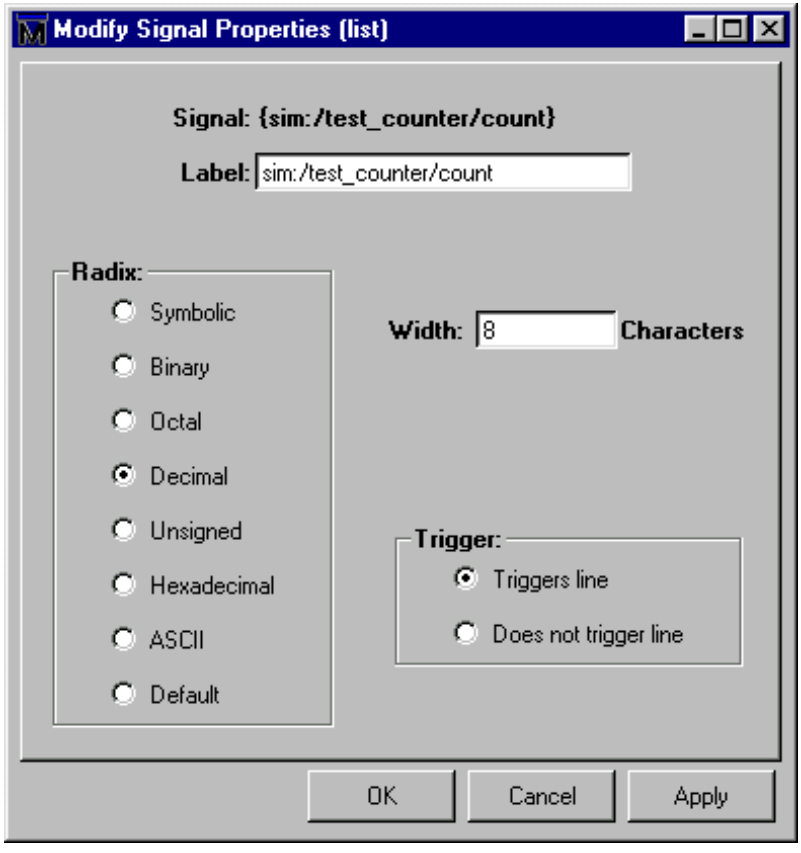
(control + c with the Main window active)



Your Source window won't look exactly like this illustration because your simulation very likely stopped at a different point.

Next we'll take a brief look at some interactive debug features of the Model*Sim* environment. To start with, let's see what we can do about the way the List window presents its data.

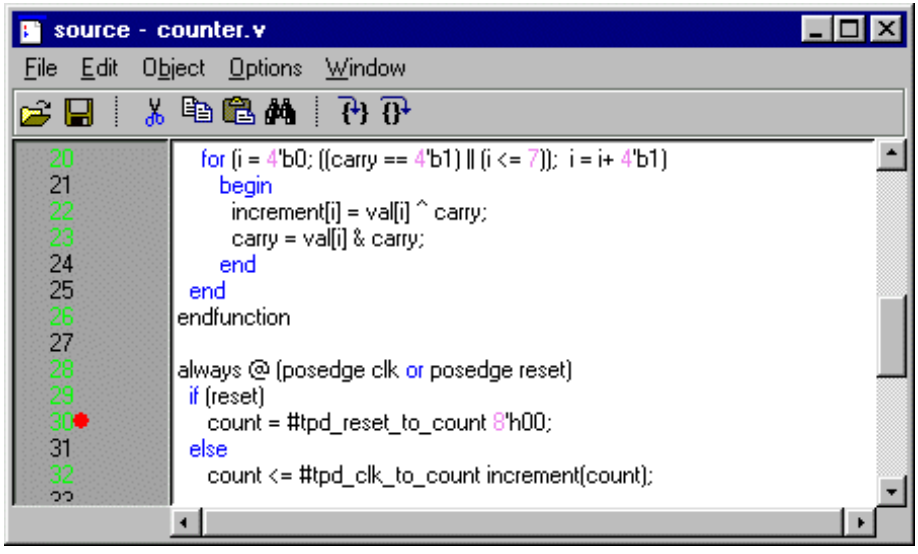
20 In the List window select **/test_counter/count**. From the List window menu bar select **Prop > Signal Props**. The Modify Signal Properties (list) dialog box is opened.



Select a display radix of **Decimal** for the signal **count**. Click **OK**. This causes the List window output to change; the count signal is now listed in decimal rather than the default binary.

21 Let's set a breakpoint at line 30 in the *counter.v* file (which contains a call to the Verilog function increment). To do this, select **dut: counter** in the Structure window. Move the cursor to the Source window and scroll the window to display line 30. Click at or near line number 30 to set a breakpoint.

Note: Breakpoints can only be used with executable lines. Executable line numbers are displayed in green. The breakpoint appears as a red dot.



22 Select the **Run -all** button from the Main toolbar to resume execution of the simulation.

(PROMPT: run -all) (Main MENU: Run -All)

When the simulation hits the breakpoint, it stops running, highlights the Source window with an arrow, and issues a Break message in the Main window.

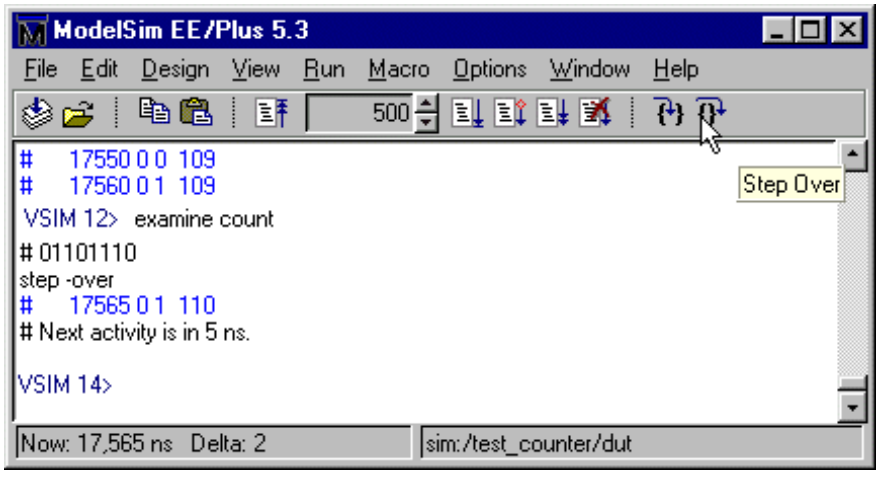
23 Typically when a breakpoint is reached you will be interested in one or more signal values. You have several options for checking values.

You can look at the values shown in the Signals window, you can move you mouse pointer over the *count* variable in the Source window and press the right mouse button, or you can use the **examine** command:

```
examine count
```

As a result of your command the count is output to the Main window.

24 Let's move through the Verilog source functions with Model*Sim*'s Step and Step Over commands. Click **Step Over** on the toolbar.



This causes the debugger to step over the function call on line 30. The Step button on the toolbar would have single-stepped the debugger, including each line of the increment function.

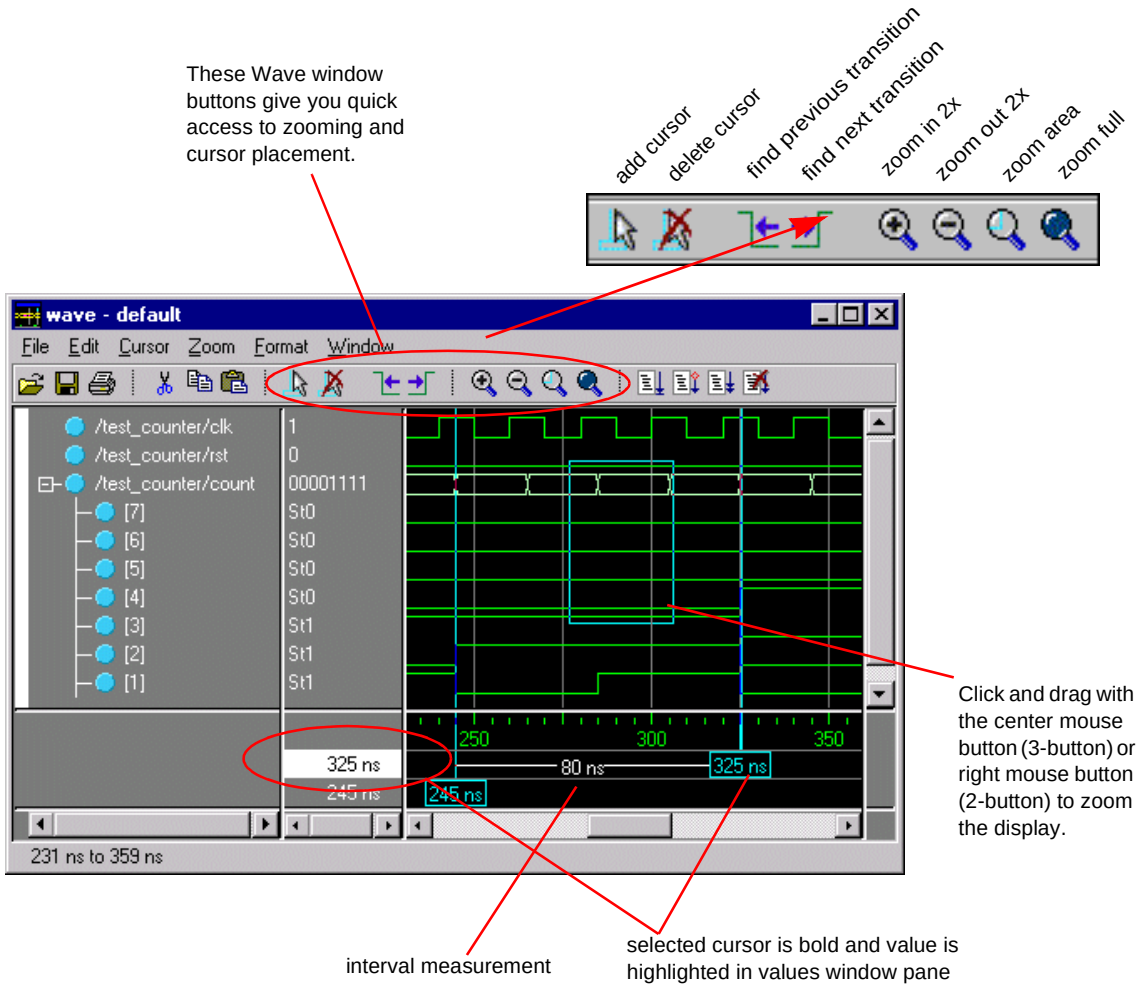
25 Experiment by yourself for awhile; setting and clearing breakpoints as well as Step'ing and Step Over'ing function calls until you feel comfortable with the operation of these commands.

26 Now let's get a Wave window view of the simulation.

When the Wave window is first drawn, there is one cursor in it at time zero. Clicking anywhere in the waveform display brings that cursor to the mouse location.

Up to twenty cursors can be present at the same time. Cursors are displayed with a time box showing the precise simulation time at the bottom. When you have more than one cursor, each time box appears in a separate track at the bottom of the display.

27 Try adding or removing cursors with the **Add Cursor** and **Delete Cursor** buttons.



When you add a cursor, it is drawn in the middle of the display. Once you have more than one cursor, VSIM adds a delta measurement showing the time difference between the two cursor positions. The selected cursor is drawn as a solid line; all other cursors are drawn with dotted lines.

28 Click in the waveform display. Notice how the cursor closest to the mouse position is selected and then moved to the mouse position.

Another way to position multiple cursors is to use the mouse in the time box tracks at the bottom of the display. Clicking anywhere in a track selects that cursor and brings it to the mouse position.

The cursors are designed to snap to the closest wave edge to the left of the mouse pointer. You can position a cursor without snapping by dragging in the area below the waveforms.

29 Experiment with using the cursors, scrolling, and zooming.

30 When you're done experimenting, quit the simulator by entering the command:

```
quit -force
```

Lesson 8 - Mixed VHDL/Verilog simulation

You must be using Model*Sim* /PLUS for this lesson.

The goals for this lesson are:

- Compile multiple VHDL and Verilog files.
- Simulate a mixed VHDL and Verilog design.
- List VHDL signals and Verilog nets and registers.
- View the design in the Structure window.
- View the HDL source code in the Source window.

1 First, return to the directory you created in Lesson 2 -Basic VHDL simulation.

```
cd <directory_name>
```

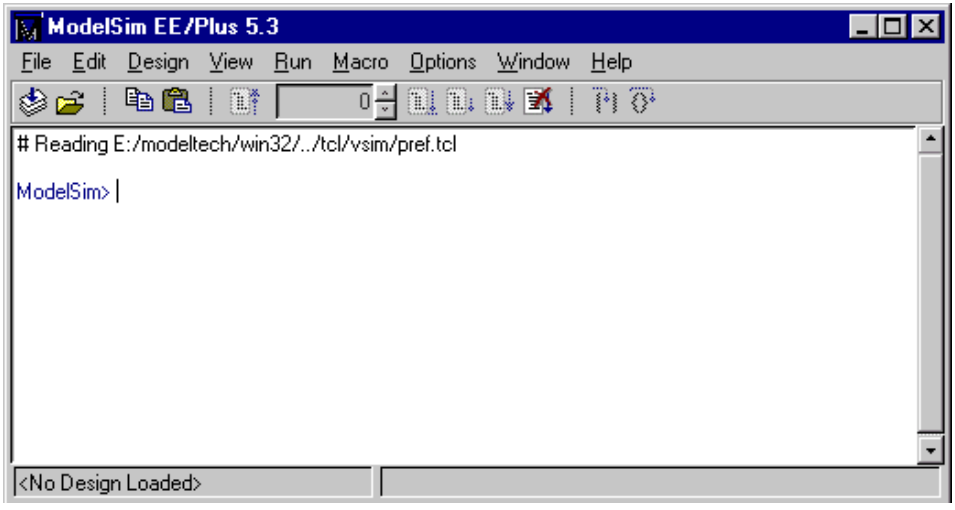
Next copy the VHDL and Verilog example files to the directory:

```
<install_dir>\modeltech\examples\mixedHDL\*.vhd
<install_dir>\modeltech\examples\mixedHDL\*.v
```

2 Start ModelSim with this command from the UNIX/DOS prompt (or *modelsim.exe* for Windows):

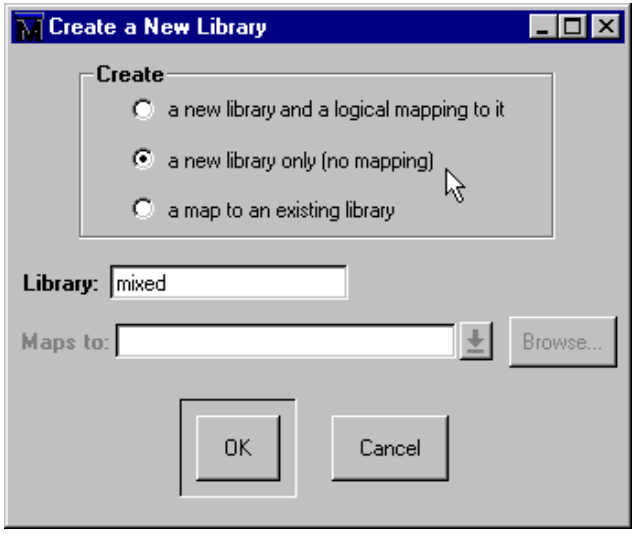
```
vsim -gui
```

This opens the Model*Sim* Main window without loading a design.



3 Let's create a new library to hold the mixed design. Make this menu selection in the Main window: **Design > Create a New Library**.

(PROMPT: vlib mixed)

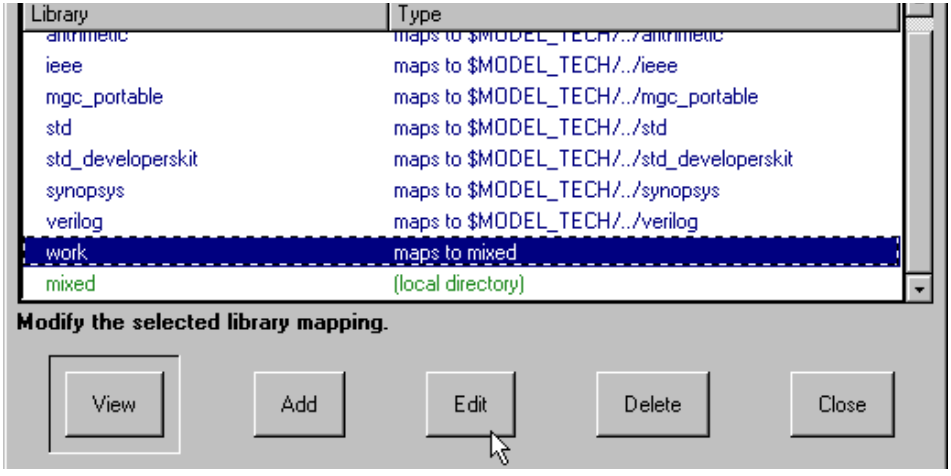


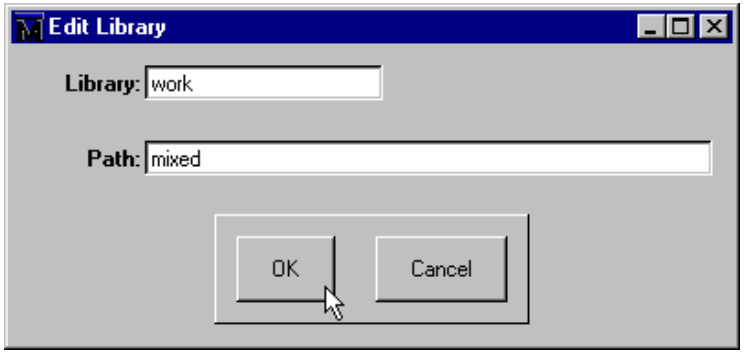
In the Create a New Library dialog box select **Create: a new library only**. In the **Library:** field type **mixed**, then select **OK**.

This creates a subdirectory named *mixed* (your design library) within the current directory. The library contains a special file named *_info* that is created with the library.

Do not create library subdirectories using UNIX or Windows commands—always use the Library menu or the **vlib** command from either the Model*Sim* or UNIX/DOS prompt.

4 Now you can map the new library to the **work** library. From the Main menu select **Design > Browse Libraries**. In the Library Browser select the **work** library, then **Edit**.
(PROMPT: vmap work mixed)





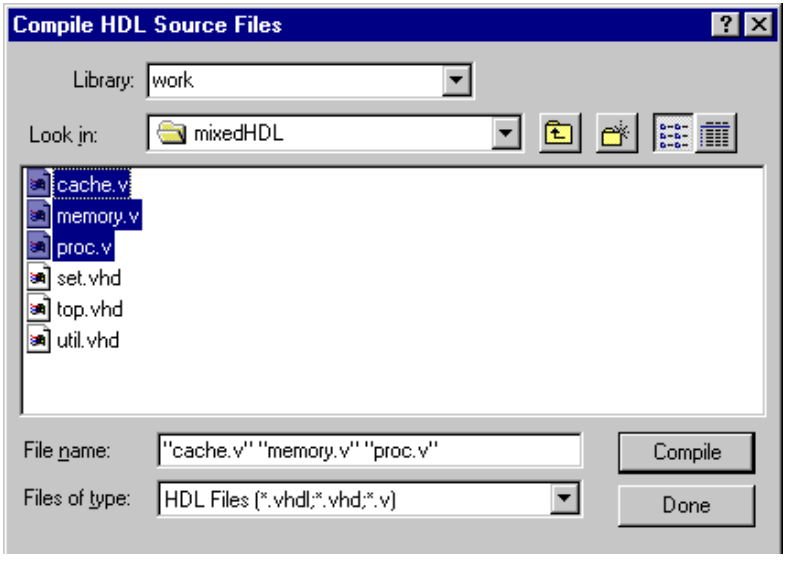
This opens the Edit Library dialog box; where you can set the library mapping between work and mixed.

Type **mixed** into the **Path:** field, then click **OK**. Now you're ready to compile the design.

5 Compile the HDL files by selecting the **Compile** button on the toolbar:

(PROMPT:vlog cache.v memory.v proc.v)
(PROMPT: vcom util.vhd set.vhd top.vhd)

This opens the Compile HDL Source Files dialog box.



A group of Verilog files can be compiled in any order. Note, however, in a mixed VHDL/Verilog design the Verilog files must be compiled before the VHDL files.

Compile the source, by double-clicking each of these Verilog files in the file list (this invokes the Verilog compiler, **vlog**):

- *cache.v*
- *memory.v*
- *proc.v*

6 Depending on the design, the compile order of VHDL files can be very specific. In the case of this lesson, the file *top.vhd* must be compiled last.

Stay in the Compile HDL Source Files dialog box and compile the VHDL files in this order (this invokes the VHDL compiler, **vcom**):

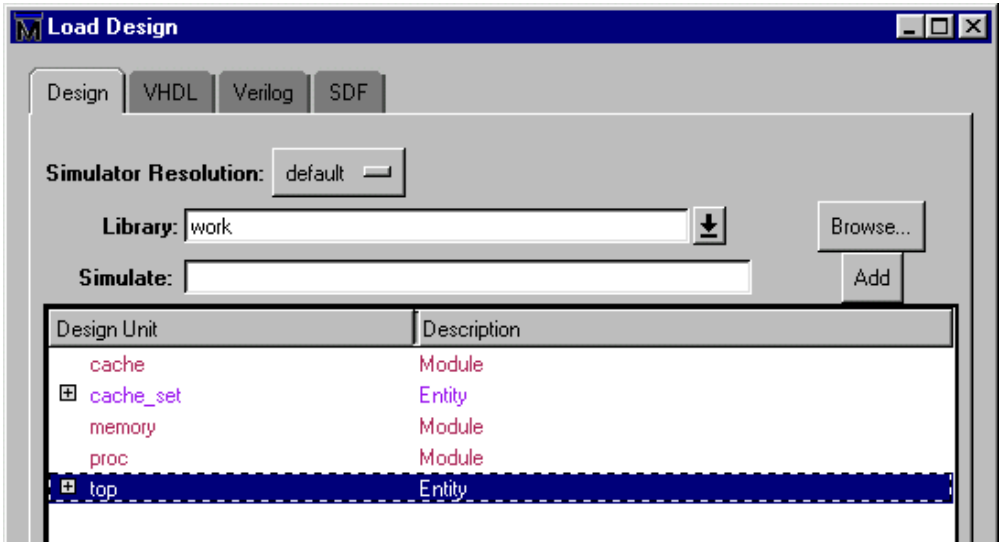
- *util.vhd*
- *set.vhd*
- *top.vhd*

Compiling is now complete, click **Done** to dismiss the dialog box.

7 Now it's time to simulate. Start the simulator by selecting the **Load Design** button from the Main toolbar:

(PROMPT: vsim top)

This returns the Load Design dialog box.



On the Design tab select the **top** entity and click **Load**.

8 From the Main menu select **View > All** to open all Model*Sim* windows.

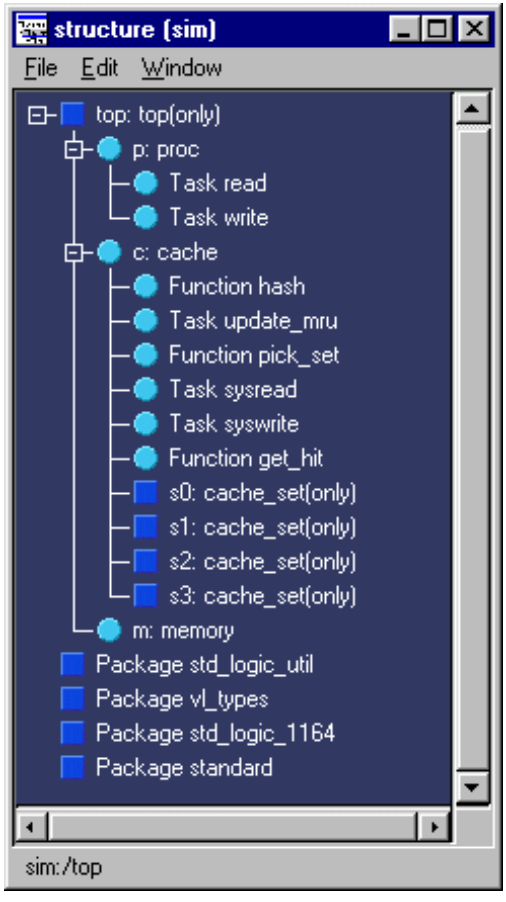
(PROMPT: view *)

9 This time you will use the VSIM command line to add all of the HDL items in the region to the List and Wave windows:

```
add list *
add wave *
```

(Signals MENU: View > List > Signals in Region)
(Signals MENU: View > Wave > Signals in Region)

10 Take a look at the Structure window.



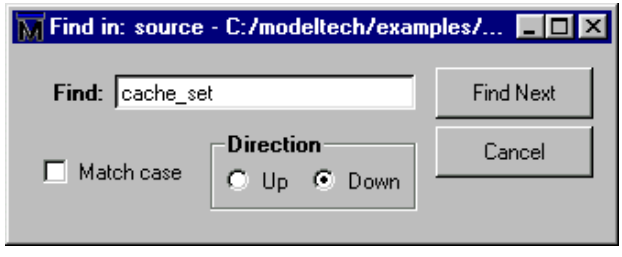
Notice the hierarchical mixture of VHDL and Verilog in the design. VHDL levels are indicated by a square "prefix", while Verilog levels are indicated by a circle "prefix." Try expanding (+) and contracting (-) the structure layers. You'll find Verilog modules have been instantiated by VHDL architectures, and similar instantiations of VHDL items by Verilog.

Let's take another look at the design.

11In the Structure window, click on the Verilog module **c: cache**. The source code for the Verilog module is now shown in the Source window.

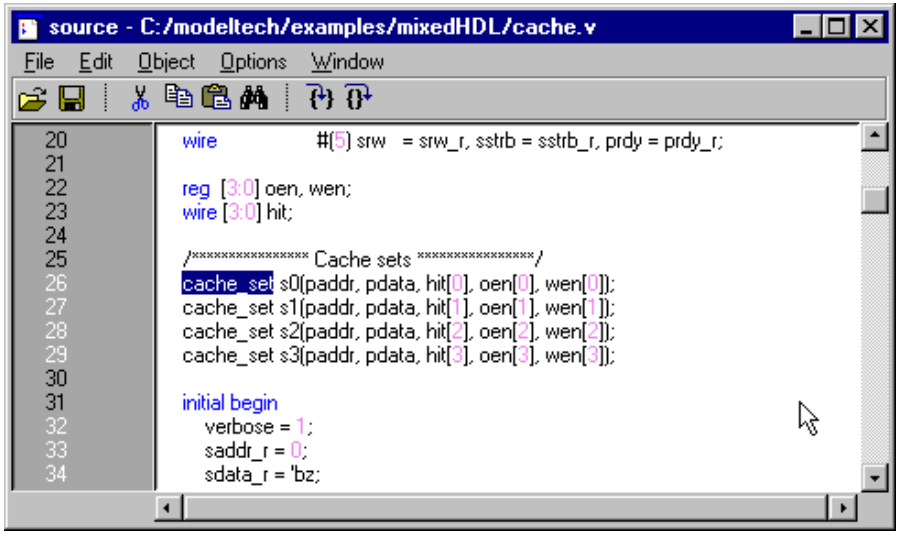
12We'll use Model*Sim*'s Find function to locate the declaration of cache_set within *cache.v*.

From the Source window menu select: **Edit > Find > Source Text**; the **Find in** dialog box is displayed.



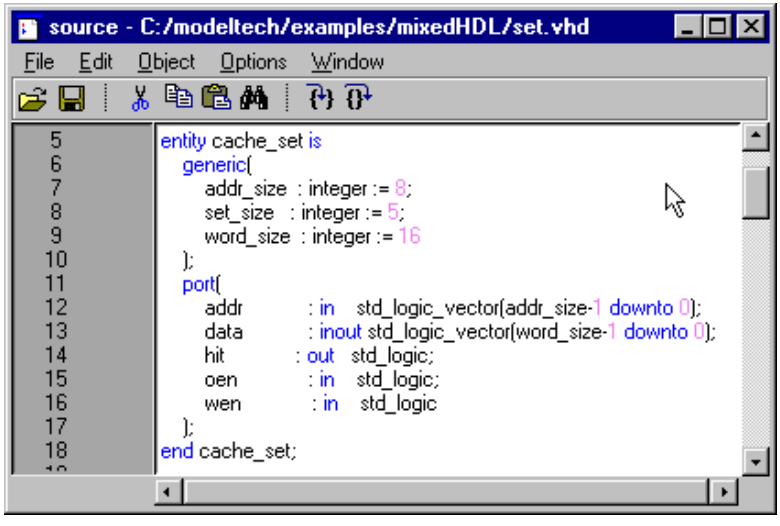
In the **Find:** field, type **cache_set** and click **Find Next**. The cache_set declaration is now displayed in the Source window. (Click **Close** to dismiss the **Find in:** dialog box.)

Note that the declaration of cache_set is a VHDL entity instantiated within the Verilog file *cache.v*.



13 Now click on the line **"s0: cache_set(only)"** in the Structure window.

The Source window now shows the VHDL code for the cache_set entity.



Before you quit, try experimenting with some of the commands you've learned from Lesson 1. Note that in this design, "clk" is already driven, so you won't need to use the **force** command.

14 When you're ready to quit simulating, enter the command:

```
quit -force
```

Lesson 9 - Simulating with Performance Analyzer

This lesson introduces the Performance Analyzer and shows you how the major Performance Analyzer commands are used. It is designed to demonstrate how the Performance Analyzer can be used to improve simulation performance. You must be using Model*Sim* EE Special Edition for this lesson.

The goals for this lesson will be to:

- Run a simulation with the Performance Analyzer OFF and note the simulation run time.
- Turn the Performance Analyzer ON and note, again, the simulation run time.
- View the Hierarchical and Ranked Profile displays.
- Use the Performance Analyzer statistics displayed in the Hierarchical Profile and the Ranked Profile to speed up simulation.

This lesson will use an example design that contains lower level VHDL blocks in the files *control.vhd*, *retrieve.vhd*, and *store.vhd*; and top level block, test bench and configuration files – *ringrtl.vhd*, *testring.vhd*, and *config_rtl.vhd*

1. Start by creating a new working directory, making it the current directory, and copying the files from *\modeltech\examples\profiler* into it.

2. Use the **vlib** command to create a **work** library in the current directory.

```
vlib work
```

(MENU: Design > Create a New Library)

3. Use the **vmap** command to map the work library to a physical directory. A *modelsim.ini* file will be written into the **work** directory.

```
vmap work work
```

4. Compile lower level blocks of the design.

```
vcom control.vhd retrieve.vhd store.vhd
```

(MENU: Design > Compile)

5. Compile top level block, test bench and configuration files.

```
vcom ringrtl.vhd testring.vhd config_rtl.vhd
```

(MENU: Design > Compile)

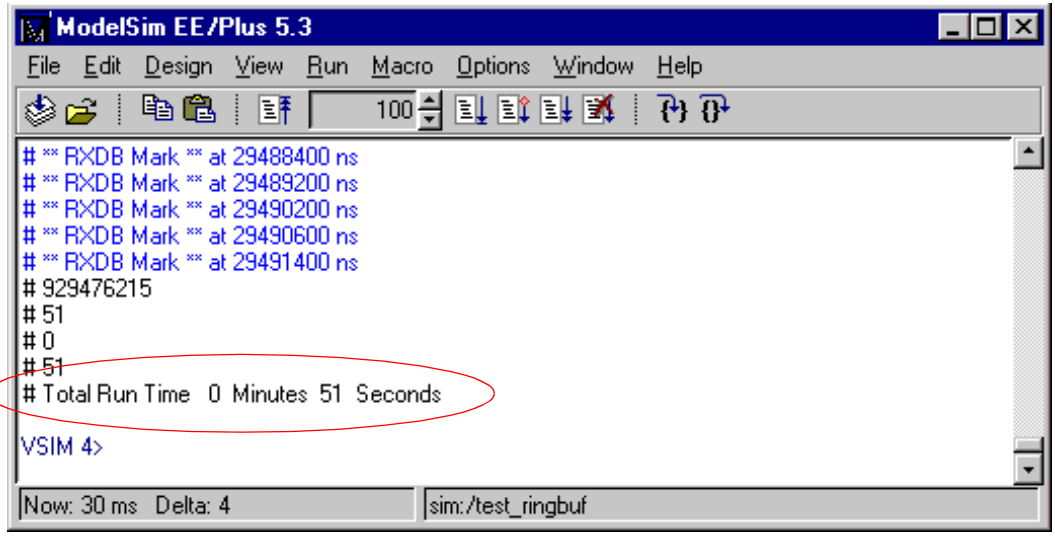
6 Use the **vsim** command to load the design configuration.

```
vsim work.test_bench_rtl
```

(MENU: Design > Load New Design)

7 Now, run the supplied DO file – *timerun.do*. This file runs the simulation and displays the total run time in the transcript area of the Main window. This test will take a minute or so.

```
do timerun.do
```



Take a look at the commands in the *timerun.do* file. The *seconds* Tcl command is used to time the simulation.

Make a note of the run time of the simulation. (Your run time will depend on the processing speed of your system and may differ from the run time shown here.)

Now we'll reset the simulation to time zero so that the simulation can be timed with the Performance Analyzer ON.

8 Restart the simulation

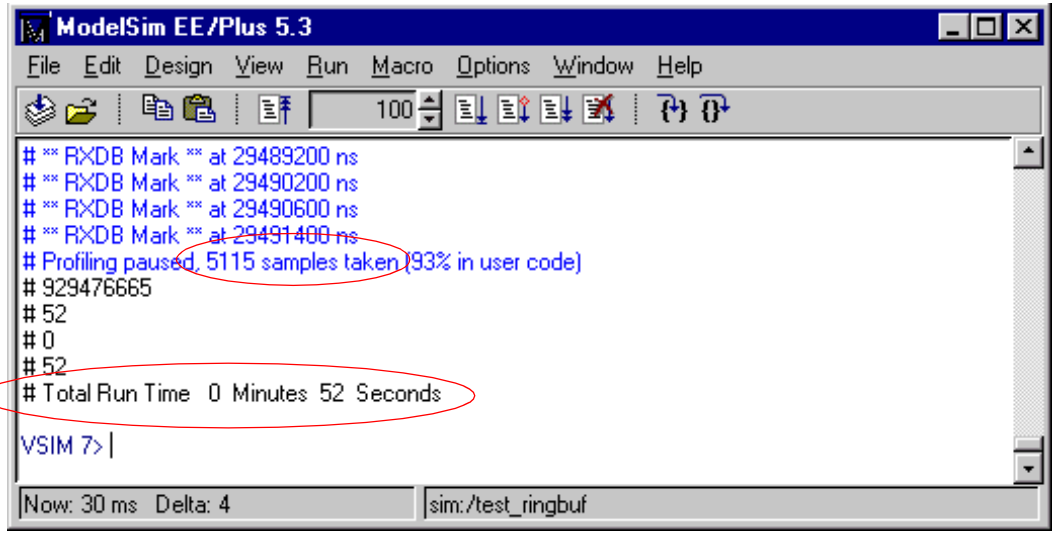
```
restart -f
```

9 Use the **profile on** command to turn on the Performance Analyzer.

```
profile on
```

10 Now use the *timerun.do* file again to run the simulation.

```
do timerun.do
```



Notice that the overhead of running the Performance Analyzer is very small (your results may differ from the results shown here), even with over 5000 samples of the simulation run acquired.

11 Display the Hierarchical Profile output.

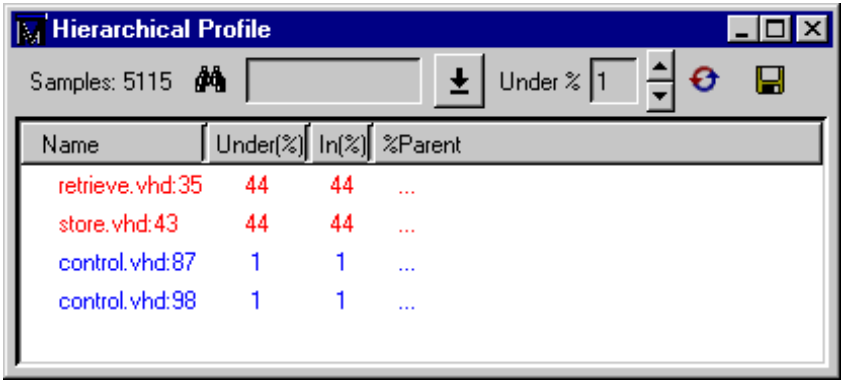
```
view_profile
```

(MENU: View > Other > Hierarchical Profile)

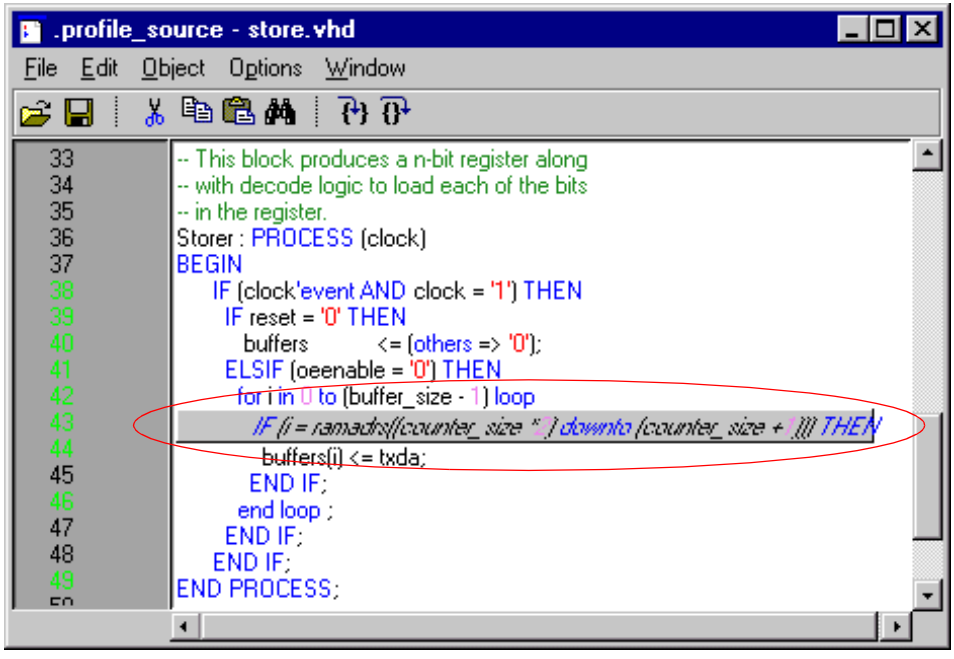
Note that two lines – *retrieve.vhd:35* and *store.vhd:43* – are taking the majority of the simulation time.

You can use the *$PrefProfile(hierCutoff)* Tcl control variable to filter out everything below a certain percentage. **hierCutoff** is the minimum percent usage that will be listed in the Hierarchical Profile display. The default value is 1%. Any usage less than 1% will not be displayed.

You can also filter the display with the Under % filter as described in step 22 below.



Double-clicking on any line in the Hierarchical Profile window will open the Source window and allow you to view the relevant source code for that line. The selected line will be highlighted in the Source window as shown below. (Here, we've double-clicked *store.vhd:43*.)



Speed up the simulation

The information provided by the Performance Analyzer can be used to speed up the simulation. Double click the pathname for *store.vhd:43* and *retrieve.vhd:35* and view the source code. In both cases, the source includes a loop which could have an exit.

12 Modify the loops to include exits inside the *IF* statements, or compile the following files included for that purpose – *store_exit.vhd* and *retrieve_exit.vhd*.

```
vcom retrieve_exit.vhd store_exit.vhd
```

(MENU: Design > Compile)

13 Compile the top level blocks and configuration files again to account for the lower level changes.

```
vcom ringrtl.vhd testring.vhd config_rtl.vhd
```

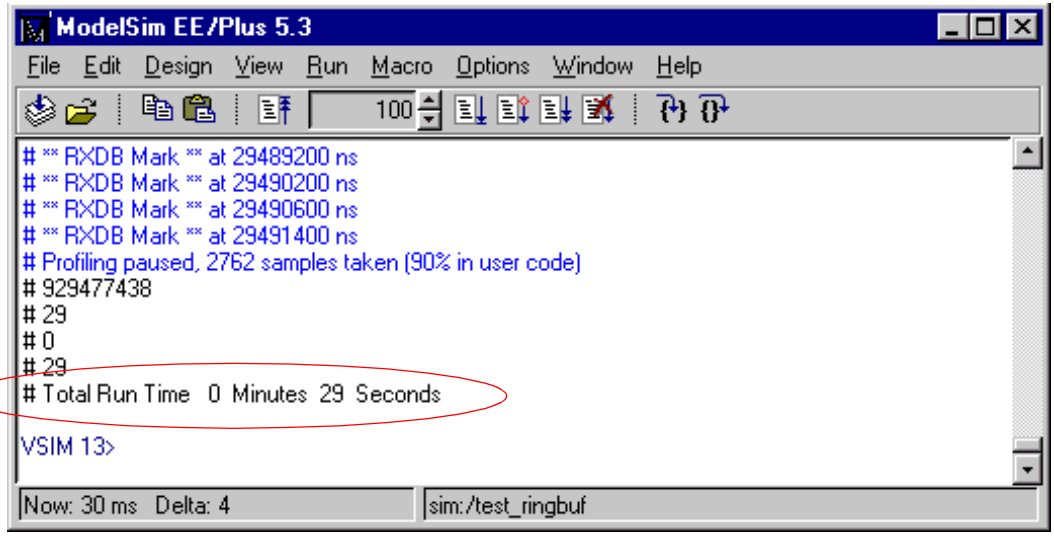
(MENU: Design > Compile)

14 Reset the simulation to time zero and restart with the modified files.

```
restart -f
```

15 Run *timerun.do* again and note the difference in run time.

```
do timerun.do
```



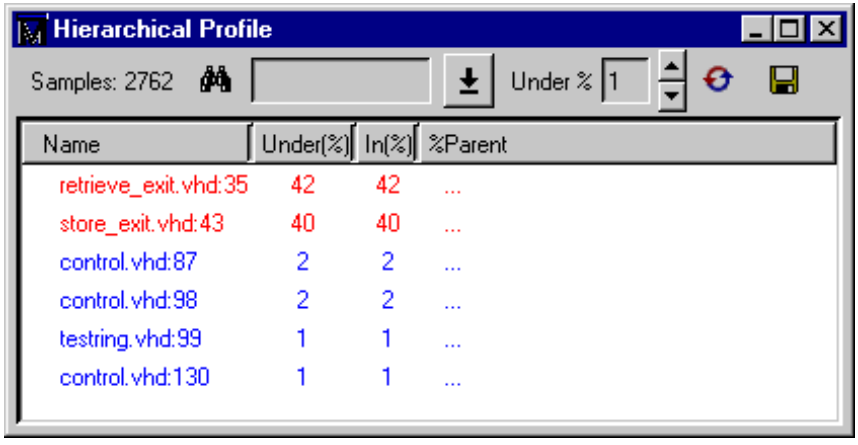
Run time has been cut almost in half by inserting exits in the loops.

16 Take another look at the Performance Analyzer data.

```
view_profile
```

(MENU: View > Other > Hierarchical Profile)

A lot of time is still being spent in the loops. To further reduce simulation time, these loops can be replaced by indexing an array.



17 Remove the loops and add an array, or compile the following files with the modifications already done.

```
vcom retrieve_array.vhd store_array.vhd
```

(MENU: Design > Compile)

18 Compile the top level blocks and configuration files again.

```
vcom ringrtl.vhd testring.vhd config_rtl.vhd
```

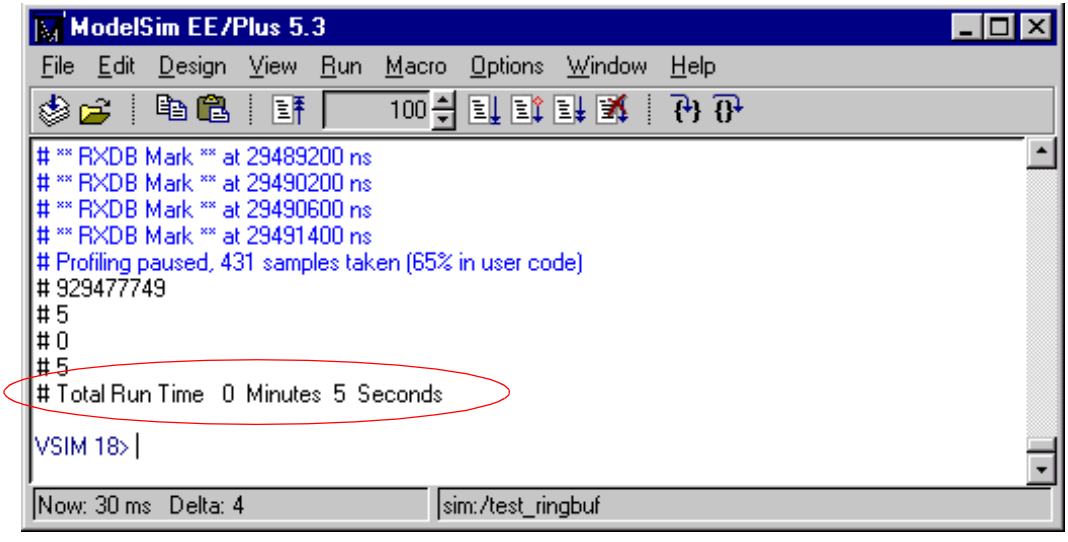
(MENU: Design > Compile)

19 Restart the simulation with the modified files.

```
restart -f
```

20 Run *timerun.do* again and note the difference in simulation run time. Your simulation time may differ from that shown here, but the new run should be very fast – approximately ten times faster than the original simulation time.

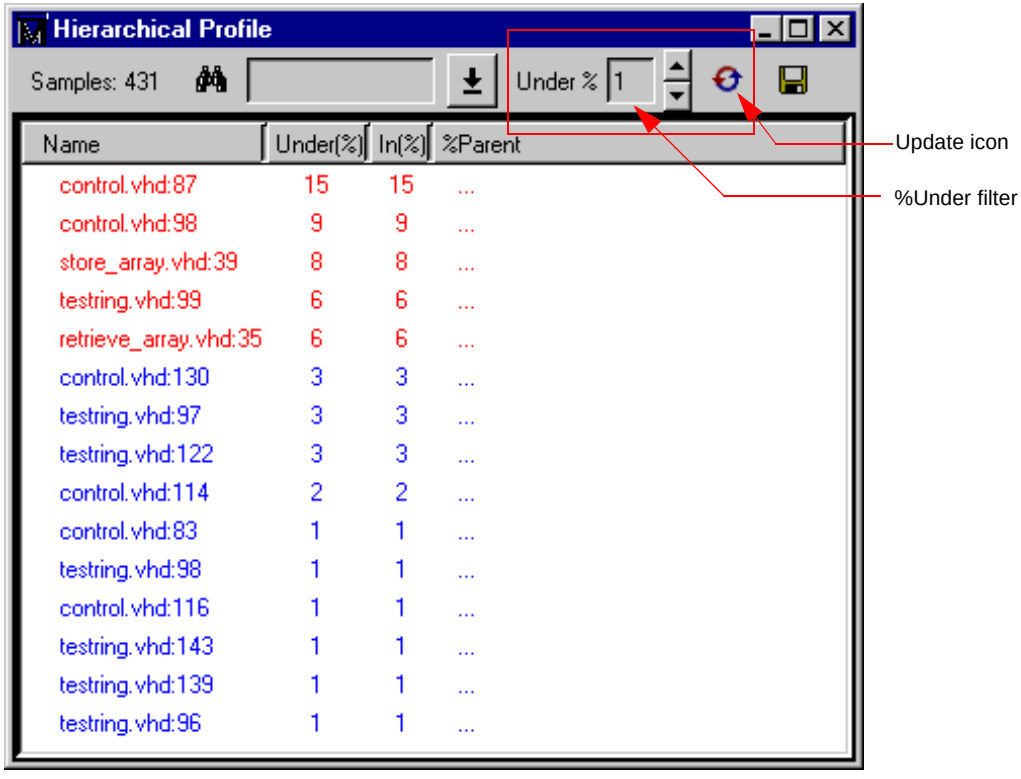
```
do timerun.do
```



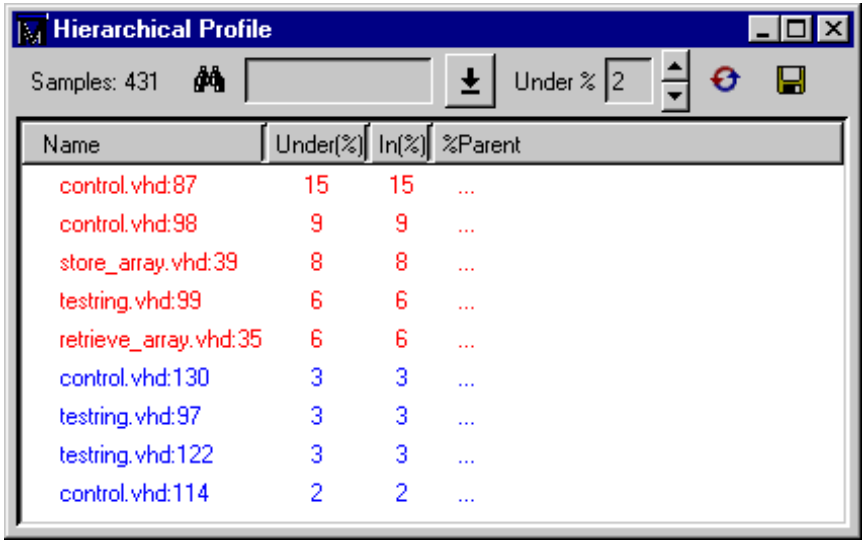
21 Look, again, at the Hierarchical Profile of simulation performance and you will see more lines showing.

```
view_profile
```

(MENU: View > Other > Hierarchical Profile)



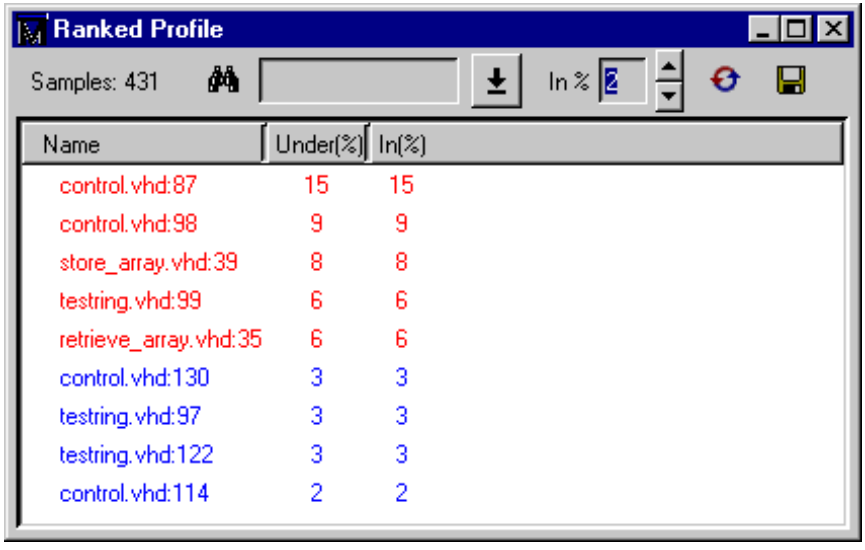
22 Set the Under% filter to "2" and click the Update icon. This will filter out all usage values below 2%.



23 Take a look at the Ranked Profile view.

```
view_profile_ranked
```

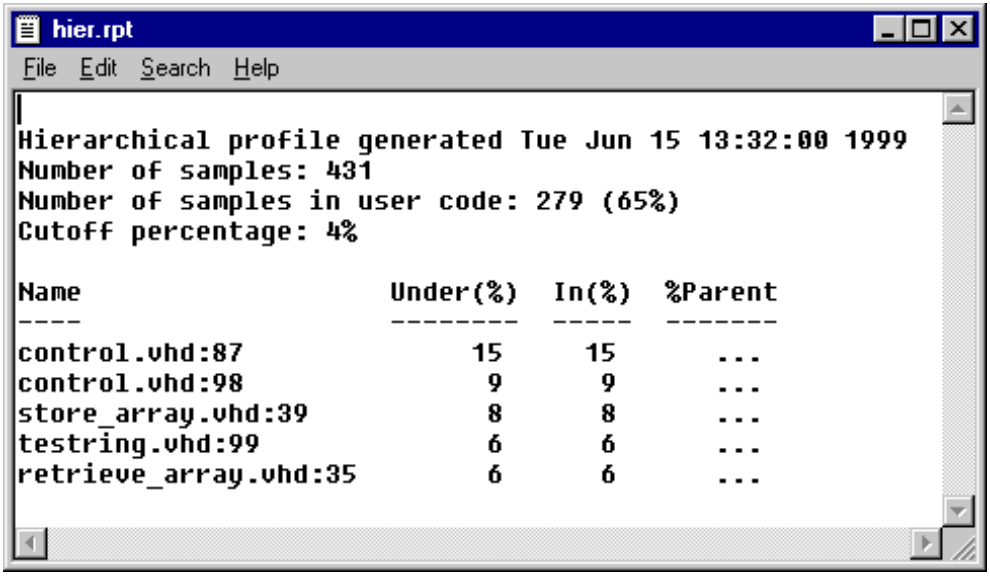
(MENU: View > Other > Ranked Profile)



24 Use the report command to output a file with the profile data.

```
profile report -hierarchical -file hier.rpt -cutoff 4
```

This command outputs a hierarchical profile of performance data with the file name *hier.rpt*.



25 Quit the simulator.

```
quit -f
```

Lesson 10 - Simulating with Code Coverage

This lesson will introduce ModelSim's Code Coverage feature, detail the use of the major Code Coverage commands, and show how to append results from more than one simulation run. In addition, it will demonstrate the small overhead associated with running code coverage. You must be using Model*Sim* EE Special Edition for this lesson.

The goals for this lesson will be to:

- Run a simulation with Code Coverage ON and examine the coverage_summary window.
- Save line coverage information to a text file.
- Append results from previous simulation run onto next one.

All commands are shown as entered on the Model*Sim* command line.

1 For this lesson, you'll use the same working directory used for the Simulating with the Performance Analyzer lesson. It is not necessary to re-create the work library or analyze the source code if you have completed the last lesson. See Lesson 8 - Mixed VHDL/Verilog simulation (p73) if you need the details on these steps.

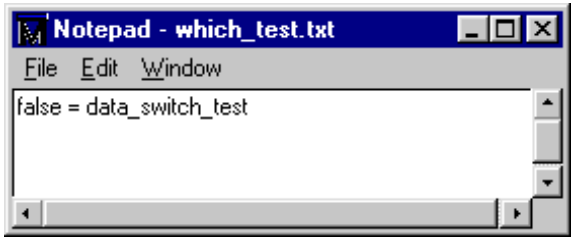
2 Compile lower level blocks of the design.

```
vcom control.vhd retrieve.vhd store.vhd
```

(MENU: Design > Compile)

3 Edit the *which_test.txt* file in the *modeltech\examples\profiler* directory to ensure it reads "false = data_switch_test". You can edit the file with **notepad** within Model*Sim*.

```
notepad which_test.txt
```



This switch configures the test bench – the *ringrtl.vhd* file. Changing this entry in the text file causes two different tests to be run from the same test bench.

4 Compile top level block, test bench and configuration files.

```
vcom ringrtl.vhd testring.vhd config_rtl.vhd
```

(MENU: Design > Compile)

5 Use the **vsim -coverage** command to load the design configuration with Code Coverage.

```
vsim -coverage work.test_bench_rtl
```

(MENU: Design > Load New Design)

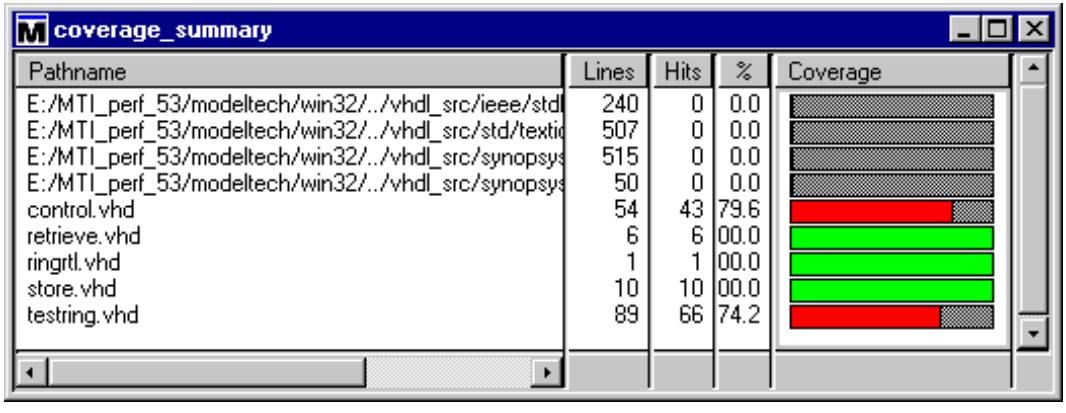
6 Run the simulator for 3 milliseconds.

```
run 3000000
```

7 Display the coverage_summary window.

```
view_coverage
```

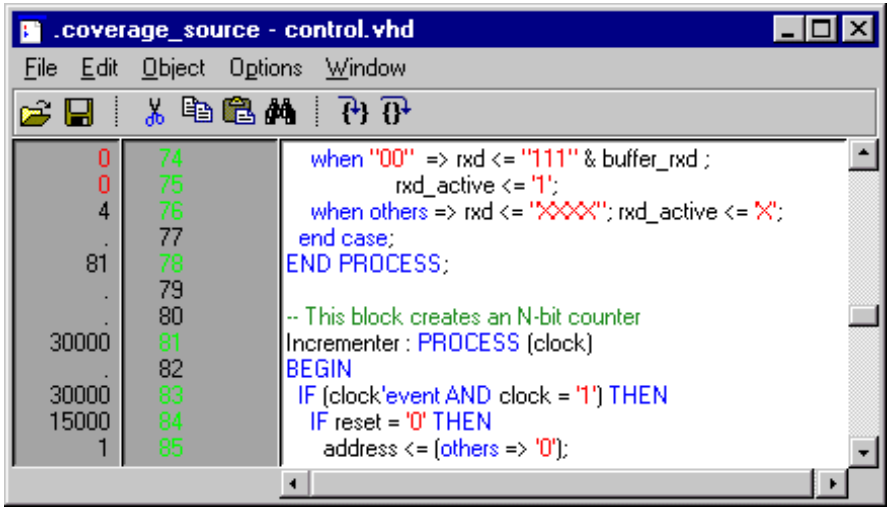
(MENU: View > Other > Source Coverage)



Note that both *testring.vhd* and *control.vhd* are below 90% and, therefore, shown in red in the Coverage bar graph. 90% is the default coverage threshold. All coverage values below 90% will be shown red. The default coverage threshold can be changed with the Tcl control variable *$PrefCoverage(cutoff)*.

8 Double-click on the *control.vhd* pathname to display the source code for *control.vhd* in the Source window. With Code Coverage enabled, the Source window is displayed with an extra column that details the number of times each line has been executed.

Scroll the Source window to view the executable lines. As you can see some lines have red zeros next to them. This indicates a line that was not executed.



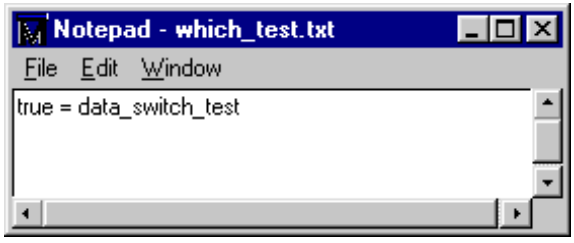
9 Save the line coverage information to a text file.

```
coverage report -file cover.dat -lines
```

Open the file *cover.dat* to see how the data is stored. Such report files can be used to help document simulation runs. **Notepad** works well to check text files such as this.

```
notepad cover.dat
```

10 Now, take note of how many times the clocked processes have been executed. Then edit the *which_test.txt file* to ensure that it reads "true = data_switch_test."



11 Restart the simulation with the changed flag so a different test is run on the circuit.

```
restart -f
```

12 Restore the coverage data from the last simulation run so that its data can be appended to the last simulation run.

```
coverage reload cover.dat
```

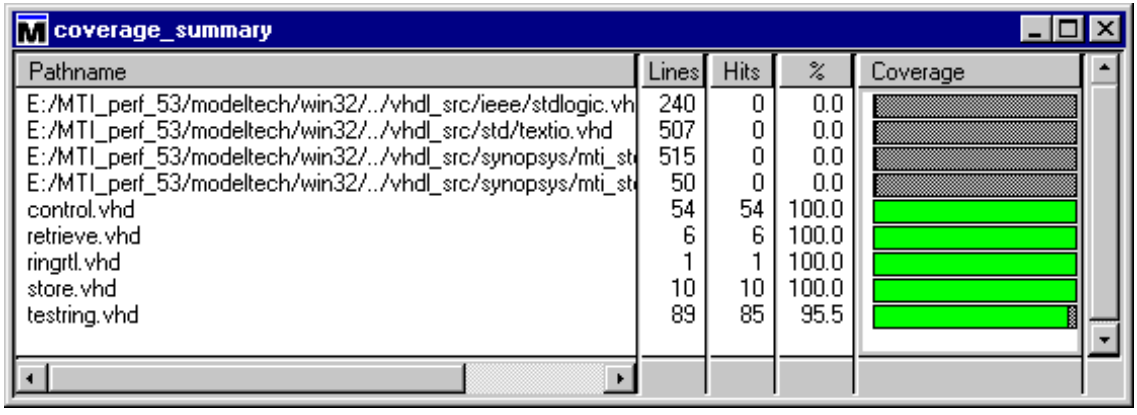
13 Run the simulator for 3msecs as before.

```
run 3000000
```

14 Display the coverage_summary window again.

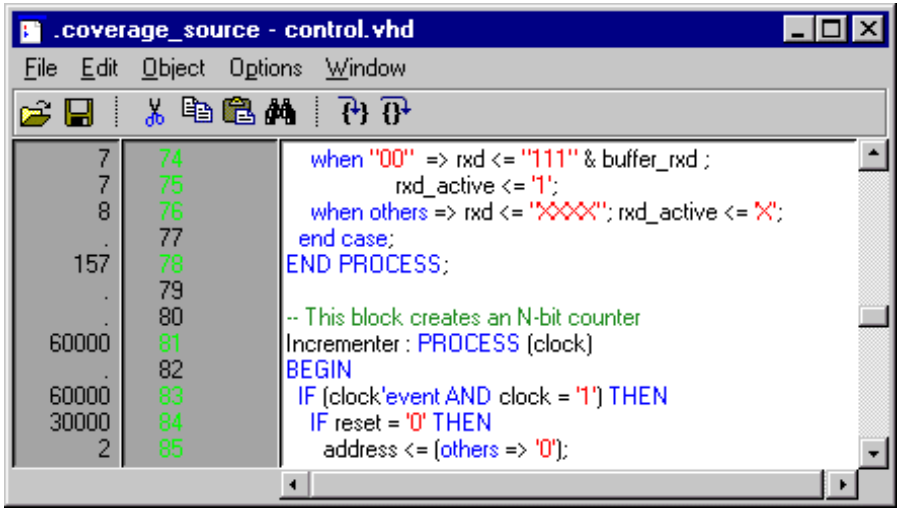
```
view_coverage
```

(MENU: View > Other > Source Coverage)



Note that now both *testring.vhd* and *control.vhd* are above 95% and therefore shown is green.

15 Double-click on the *control.vhd* pathname to bring up the Source window. You can see from the values in the first column that the line hits from this run has been added to the ones from the last run. The number of times the clocked processes have been run have doubled.



16 Compile the lower level blocks with all optimizations switched off. This will cause more executable lines to be shown, and coverage data will be collected for the packages.

```
vcom -O0 -noaccel std_logic_arith -noaccel std_logic_unsigned -noaccel
std_logic_1164 control.vhd store.vhd retrieve.vhd
```

17 Compile top level blocks and configuration with optimizations switched off.

```
vcom -O0 -noaccel textio -noaccel std_logic_arith -noaccel std_logic_unsigned -
noaccel std_logic_1164 ringrtl.vhd testring.vhd config_rtl.vhd
```

18 Restart the simulation with the modified files.

```
restart -f
```

19 Run the simulator for 3msecs as before.

```
run 3000000
```

20 Display the coverage summary window again.

```
view_coverage
```

(MENU: View > Other > Source Coverage)

Note that now line hits will be shown in the packages that were optimized. This could be used on separate files to work out which functions are used in the IEEE packages.

Take a look at the report files and other commands used to control the coverage tool. Time the simulation runs with and without coverage using the “do” file provided for the Performance Analyzer lesson. There should be little overhead running with Code Coverage enabled.

21 Quit the simulator.

```
quit -f
```

Lesson 11 - Finding names, and searching for values

The goals for this lesson will be to:

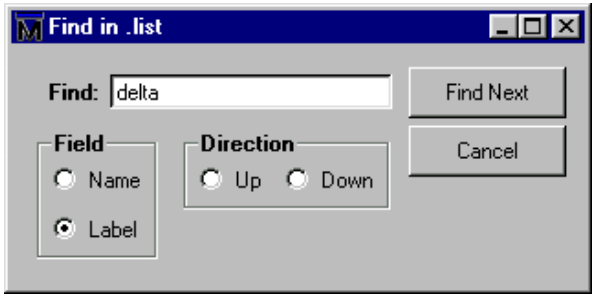
- Find items by name in tree windows.
- Search for item values in the List and Wave windows.

You can easily locate HDL item names and values within Model*Sim*'s windows. Start any of the lesson simulations to try out the Find and Search functions illustrated below.

Finding items by name in tree windows

You can find HDL item names with the **Edit > Find** menu selection in these windows:
List, Process, Signals, Source, Structure, Variables, and Wave windows.

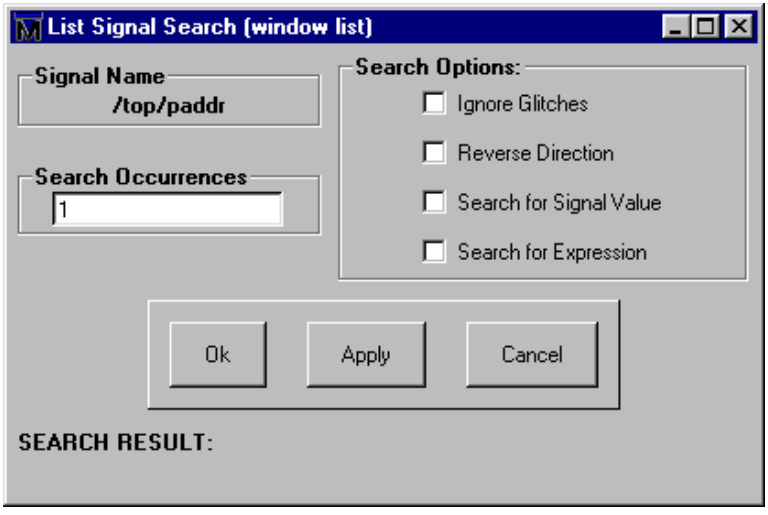
Select **Edit > Find** to bring up the Find dialog box (List window version shown).



Enter an item label and **Find** it by searching **Forward** (right) or **Reverse** (left) through the window display.

Searching for item values in the List and Wave windows

You can search for HDL item values in the List, and Wave windows. Select **Edit > Search** from the window's menu to bring up the Signal Search dialog box (List window version shown).



The **List Signal Search** dialog box includes these options:

You can locate values for the **Signal Name: <item_label>** shown at the top of the dialog box. The search is based on these options (multiple **Search Options** may be selected):

- **Search Options: Ignore Glitches**
  Ignore zero width glitches in VHDL signals and Verilog nets.
- **Search Options: Reverse Direction**
  Search the list from bottom to top.
- **Search Options: Search for Signal Value**
  Activates the **Search Value** field; search for the value specified in the **Search Value** field, otherwise just look for transitions.
- **Search Options: Search for Expression**
  Activates the **Search Expression** field and the **Use Expression Builder** button; searches for the expression specified in the Search Expression field evaluating to a boolean true.

  The expression may involve more than one signal but is limited to signals logged in the List window. Expressions may include constants, variables, and DO files. If no expression is specified, the search will give an error. See the *ModelSim Reference Manual* for more information on expression syntax and the use of the Expression Builder.

- **Search Occurrences**
  You can search for the n-th transition or the n-th match on value; **Search Occurrences** indicates the number of transitions or matches for which to search.
- **Search Value**
  Valid only if **Use signal value** is selected; specifies the search value; must be formatted in the same radix as displayed.

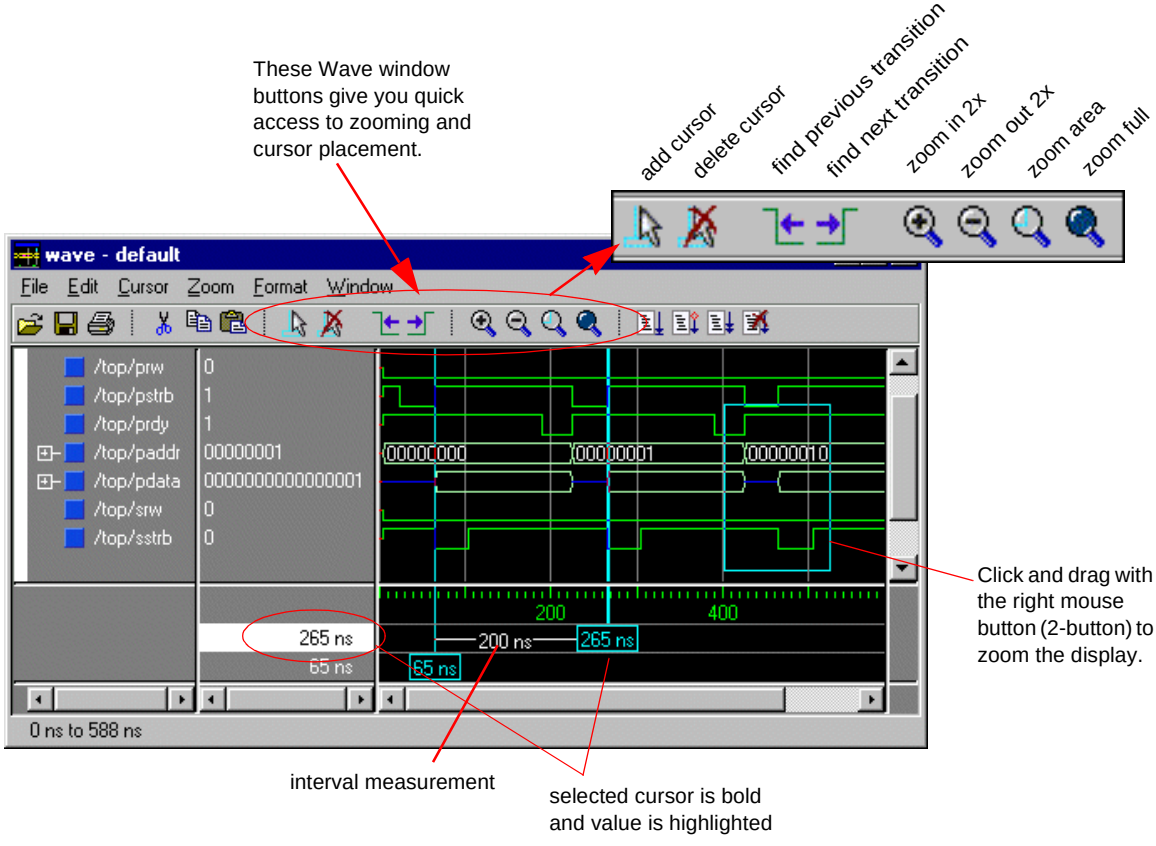
The result of your search is indicated at the bottom of the dialog box.

Lesson 12 - Using the Wave window

The goals for this lesson are:

- Practice use of Wave window time cursors.
- Practice the different methods for zooming the waveform display.
- Practice using Wave window keyboard shortcuts.
- Practice combining items into a virtual object.

Any of the previous lesson simulations may be used with this practice, or you can use one of your own simulations if you wish.



Using time cursors in the Wave window

When the Wave window is first drawn, there is one cursor located at time zero. Clicking anywhere in the waveform display brings that cursor to the mouse location.You can add additional cursors to the waveform pane with the **Cursor > Add Cursor** menu selection (or the Add Cursor button shown below). The selected cursor is drawn as a bold solid line; all other cursors are drawn with thin solid lines. Remove cursors by selecting them and choosing using the **Cursor > Delete Cursor** menu selection (or the Delete Cursor button shown below).

Add Cursor add a cursor to the center of the waveform window	**Delete Cursor** delete the selected cursor from the window

Finding a cursor

The cursor value (on the **Goto** list) corresponds to the simulation time of that cursor. Choose a specific cursor view with **Cursor > Goto** menu selection.

Making cursor measurements

Each cursor is displayed with a time box showing the precise simulation time at the bottom. When you have more than one cursor, each time box appears in a separate track at the bottom of the display. VSIM also adds a delta measurement showing the time difference between the two cursor positions.

If you click in the waveform display, the cursor closest to the mouse position is selected and then moved to the mouse position. Another way to position multiple cursors is to use the mouse in the time box tracks at the bottom of the display. Clicking anywhere in a track selects that cursor and brings it to the mouse position.

The cursors are designed to snap to the closest wave edge to the left on the waveform that the mouse pointer is positioned over. You can control the snap distance from "Wave category" in the dialog box available from the Wave window **Prop > Display Props** menu selection.

You can position a cursor without snapping by dragging in the area below the waveforms.

You can also move cursors to the next transition of a signal with these toolbar buttons:

Find Previous Transition locate the previous signal value change for the selected signal	**Find Next Transition** locate the next signal value change for the selected signal

Zooming - changing the waveform display range

Zooming lets you change the simulation range in the windowpane display. You can zoom with either the **Zoom** menu, toolbar buttons, mouse, keyboard, or VSIM commands.

Using the Zoom menu

You can use the Wave window menu bar, or call up the **Zoom** menu by clicking the right mouse button (of a three-button mouse) in the right windowpane.

Note: The right mouse button of a two-button mouse will not open the **Zoom** menu. It will, however, allow you to create a zoom area by dragging left to right while holding down the button.

The Zoom menu options include:

- **Zoom Full**
  Redraws the display to show the entire simulation from time 0 to the current simulation time.
- **Zoom In**
  Zooms in by a factor of two, increasing the resolution and decreasing the visible range horizontally, cropping the view on the right. The starting time is held static.
- **Zoom Out**
  Zooms out by a factor of two, decreasing the resolution and increasing the visible range horizontally, extending the view on the right. The starting time is held static.
- **Zoom Last**
  Restores the display to where it was before the last zoom operation.
- **Zoom Area with Mouse Button 1**
  Use mouse button 1 to create a zoom area. Position the mouse cursor to the left

side of the desired zoom interval, press mouse button 1 and drag to the right. Release when the box has expanded to the right side of the desired zoom interval.

- **Zoom Range**
  Brings up a dialog box that allows you to enter the beginning and ending times for a range of time units to be displayed.

Zooming with the toolbar buttons

These zoom buttons are available on the toolbar:

Zoom in 2x zoom in by a factor of two from the current view	**Zoom area** use the cursor to outline a zoom area
Zoom out 2x zoom out by a factor of two from current view	**Zoom Full** zoom out to view the full range of the simulation from time 0 to the current time

Zooming with the mouse

To zoom with the mouse, position the mouse cursor to the left side of the desired zoom interval, press the middle mouse button (three-button mouse), or right button (two-button mouse), and while continuing to press, drag to the right and then release at the right side of the desired zoom interval.

Zooming keyboard shortcuts

See "Wave window keyboard shortcuts" (p107) for a complete list of Wave window keyboard shortcuts.

Zooming with VSIM commands

The **.wave.tree zoomfull** provides the same function as the **Zoom > Zoom Full** menu selection and the **.wave.tree zoomrange** provides the same function as the **Zoom > Zoom Range** menu selection.

Use this syntax with the **.wave.tree zoomrange** command:

Syntax

```
.wave.tree zoomrange f1 f2
```

Arguments

`f1 f2`
Sets the waveform display to zoom from time f1 to f2, where f1 and f2 are floating point numbers.

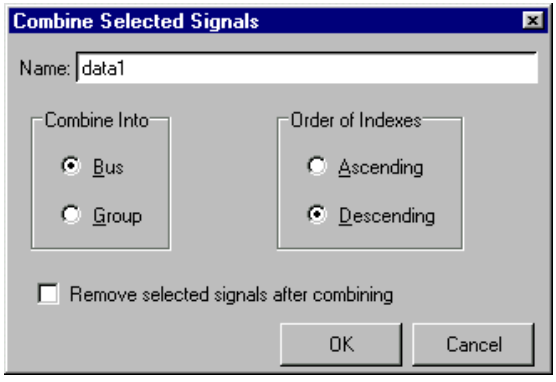
Wave window keyboard shortcuts

Using the following keys when the mouse cursor is within the Wave window will cause the indicated actions:

Key	Action
i I or +	zoom in
o O or -	zoom out
f or F	zoom full
l or L	zoom last
r or R	zoom range
<arrow up>	scroll waveform display up
<arrow down>	scroll waveform display down
<arrow left>	scroll waveform display left
<arrow right>	scroll waveform display right
<page up>	scroll waveform display up by page
<page down>	scroll waveform display down by page
<tab>	searches forward (right) to the next transition on the selected signal
<shift-tab>	searches backward (left) to the previous transition on the selected signal
<Control-f>	opens the find dialog box; search within the specified field in the wave-name pane for text strings

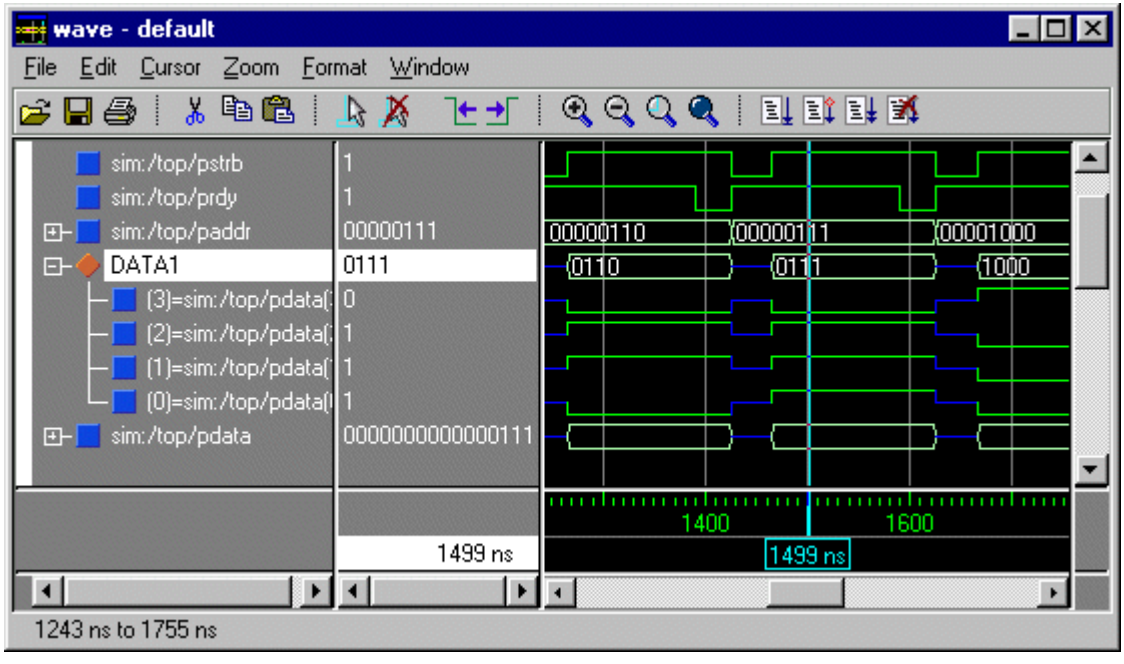
Combining and grouping items in the Wave window

The Wave window allows you to combine signals into buses or groups. Use the **Edit** > **Combine** menu selections to call up the Combine Selected Signals Dialog box.

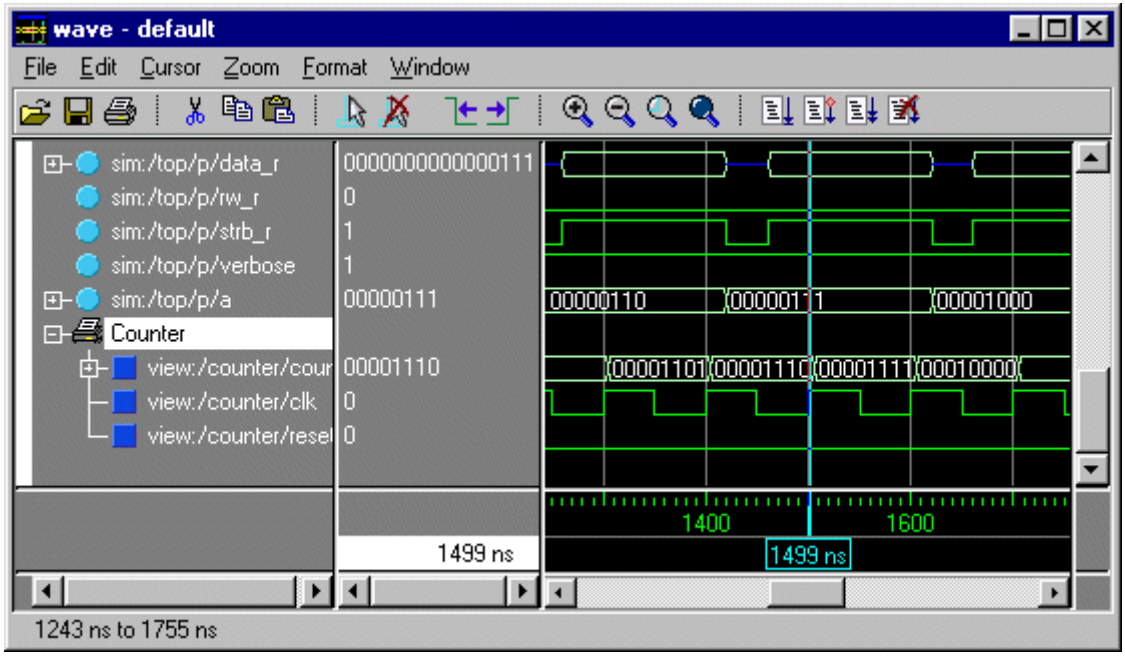


A bus is a collection of signals concatenated in a specific order to create a new virtual signal with a specific value.

In the illustration below, four data signals have been combined to form a new bus called DATA1. Notice, the new bus has a value that is made up of the values of its component signals arranged in a specific order. Virtual objects are indicated by an orange diamond.



A group is simply a container for any number of signals. It has no value, and the signals contained within it may be arranged in any order. In the illustration below, the signals counter/count, counter/clk, and counter/reset have been combined in a group called Counter. Notice that the Counter group has no value associated with it. The counter, clk and reset signals may be arranged in any order.



Adding Dividers

Using the **File > New Divider** menu selection you can also add divders to the active window pane.

Index

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A

Assertion errors 33

B

Batch-mode simulation 41
Breakpoints 27
    continuing simulation after 28

C

Code Coverage 93
    coverage_summary window 94
    reload 96
    report 95
    vsim -coverage command 94
Command history 11
Compile
    compile order 76
    compile order of Verilog modules 62
    mixed HDL design 75
    projects 17
    Verilog 59
coverage_summary window 94
Cursors 29
    find signal transitions 29
    measure time intervals 29
    probe for values 29

D

Debugging a VHDL design 31
Design
    browse libraries 75
    create new library 74
Design library
    create new 60
    creating 21
do command 12
DO files
    executing a DO file in batch-mode 41
    using a DO file at startup 43
    using the transcript as a DO file
documentation 9
drag and drop 11

E

Edit
    projects 17
Email
    Model Technology's email address 9
Errors
    breaking on assertion 34
    finding in VHDL designs 33
    viewing in Source window 35
examine command 69
examples
    Tcl example solutions 49

F

Find dialog box 99
Finding
    a cursor in the Wave window 104
Finding names, and searching for values in windows 99
force command 25

H

Hierarchical Profile 84
    update icon 89
Hierarchy
    of a mixed VHDL/Verilog design 78

of a Verilog design 65
Home page
Model Technology's home-page URL 9

I

Initialization file, see Project files

K

Keyboard shortcuts, Wave window 107

L

Libraries
creation and mapping 31
logical mapping 60
List window
change display radix 67
placing top level Verilog signals in 63
Load Design 76
Load design 18, 24

M

Macros
see Do files
modelsim.ini, see Project files
.mpf file extension 14

O

Operating systems supported 8

P

Performance Analyzer 81
hierarchical profile 84
ranked profile 90
report command 91
profile on command 83
Project files
compile 17
create 14
creation wizard 15
edit 17

Q

Quick Start
menus 13
quit VSIM command 30, 72

R

Ranked Profile 90
report command 91
Reusing commands 12
Run length selector 26
change run length 33
run VSIM command 26

S

Search options 100
Searching
for HDL item names and transitions in the Wave window 105
for values and finding names in windows 99
in tree windows 99
Shortcuts
command history 11
Wave window 107
Signal transitions
searching for 105
Signals
add to List window 32
add to Wave window 33

applying stimulus to 25
display values with examine command 69
listing in region 25
placing top-level Verilog signals in the List and Wave window 63
specifying radix of 38
triggering listings for 37
Simulating
code coverage 93
load design dialog box 18
with Performance Analyzer 81
Simulation
batch-mode 41
executing commands at startup 43
Load Design dialog box 62
mixed VHDL/Verilog 73
saving results in log file 42
single-stepping 28
starting 32
Verilog 59
-view switch 42
-wav switch 42
solutions to the examples 49
Standards supported 8
System initialization file 43

T

Tcl/Tk
how it works with ModelSim 45
references 45
Tcl source command 50
Tk widgets 49
Transcript
save 12
transcript DO file 51
Triggering
changing in List window 37
modify 38

V

Verilog
compile 59
interface checking between design units 62
viewing design in Structure and Source windows 78
Verilog simulation 59
view_profile command 84
vsim -coverage command 94

W

Wave window
cursors 70
placing top level Verilog signals in 63
Windows
finding HDL item names 99
searching for HDL item values 99
List window
locating time markers 99
viewing all 32
Wave window
changing display range (zoom) 105
cursor measurements 104
locating time cursors 99
using time cursors 104
zooming 105
Work library mapping 31

Z

Zoom
from Wave toolbar buttons 106
from Zoom menu 105
with the mouse 106
Zooming in the Wave window 105